On Byzantine Generals with Alternative Plans

Miguel Correia *, Alysson Neves Bessani, Paulo Veríssimo

Universidade de Lisboa, Faculdade de Ciências, LASIGE

Campo Grande, Bloco C6, Piso 3, 1749-016 Lisboa, Portugal

Abstract

This paper proposes a variation of the Byzantine generals problem (or Byzantine consensus). Each general has a set of good plans and a set of bad plans. The problem is to make all loyal generals agree on a good plan proposed by a loyal general, and never on a bad plan.

Key words: distributed computing, fault tolerance, Byzantine faults, agreement, consensus

1 Introduction

The *Byzantine generals problem* is one of the most well-known problems in distributed systems (Lamport et al., 1982). The idea is to make a set of loyal Byzantine generals (distributed processes) agree on a plan to attack an enemy city or retreat (a binary value). The decision must be unanimous among loyal generals, but there can be some traitors that try to break this unanimity. Solutions for this problem, later called Byzantine agreement or binary consensus, have been shown to be sufficient to solve other forms of consensus (e.g. (Correia et al., 2006)) and consensus has been show to be equivalent to many important distributed systems problems (e.g. (Hadzilacos and Toueg, 1994)).

* Corresponding author. Phone +351 217500125. Fax +351 217500084.
Email addresses: mpc@di.fc.ul.pt (Miguel Correia), bessani@di.fc.ul.pt (Alysson Neves Bessani), pjv@di.fc.ul.pt (Paulo Veríssimo).

This paper proposes a variation of the Byzantine generals problem, which we call the *Byzantine Generals with Alternative Plans* problem (BGAP). Suppose we have generals that are "smarter" than the original ones. They can devise several *good plans*, instead of only attack/retreat, and they can also devise a set of *bad plans*. The problem is to make all loyal generals agree on a good plan proposed by a loyal general. There are four flavors of the problem that depend on the creativity and technical knowledge we assume for the generals: loyal generals may have equal or different sets of good plans and equal or different sets of bad plans.

Our interest in this problem is not merely theoretical. On the contrary: the purpose is to provide an expressive abstraction that simplifies the solution of problems that involve Byzantine agreement and that would be harder to solve using other flavors of consensus, like multi-valued consensus. The BGAP problem appeared recently when we were designing Byzantine fault-tolerant tuple spaces (Bessani et al., 2006b, 2007, 2008), using both Byzantine quorum systems (Malkhi and Reiter, 1998) and state machine replication (Schneider, 1990). A *tuple space* is a shared memory object that provides operations for storing and retrieving ordered data sets called *tuples* (finite sequences of values) (Gelernter, 1985). When a process wants to read or remove a tuple from the tuple space it provides a *template* and one of the tuples in the space that *matches* the template is returned. Therefore, a tuple space is a sort of (shared) associative memory: tuples are accessed through their contents, not through their address. The original motivation for defining and studding the BGAP problem was that when a process wants to remove a tuple, the servers (Byzantine generals) have to choose a tuple in the space (a good plan) and never a tuple that has already been removed from the space (a bad plan).

We believe the BGAP problem is usually the problem we want solved for the coordination of distributed processes that can fail in a Byzantine way. The problem is quite generic: each process can have several initial values (in extreme cases, only one or all), and also a set of values it can not accept to be the decision. The notions of good and bad plans reflect the idea that processes may have several views

about what to do in a certain situation, but also that some actions are unacceptable. Therefore, many coordination problems can be solved using an algorithm that solves BGAP. In a sense BGAP tackles a limitation of other variations of the problem. Many distributed applications require that the plan decided is one of those proposed by loyal generals, while in practice most algorithms can only ensure that in special cases (if plans are binary, if all loyal generals propose the same plan).

For clarity, from now on we present the problem in terms of distributed systems and processes, not in terms of battles and Byzantine generals.

2 System Model

The system is composed by a set of n processes $\Pi = \{p_1, p_2, ...p_n\}$. The processes communicate by passing messages. A process is said to be *correct* if it follows its algorithm. Otherwise it is said to be *faulty*. We assume at most $t = \lfloor \frac{n-1}{3} \rfloor$ processes can fail, so $n \geq 3t + 1$. These failures can be Byzantine, meaning that faulty processes can stop, omit messages, send incorrect messages, send several messages with the same identifier, etc., either alone or in collusion with other faulty processes. We use the predicate $correct(i)$ in logic expressions, to say that process p_i is correct. Π, n and t are known by all processes in Π.

The system is *asynchronous*, which means that there are no bounds on the processing times or communication delays [1] . Consensus has been proved to have no deterministic solution in an asynchronous system if even a single process is allowed to stop (Fischer et al., 1985). Therefore we also assume that the asynchronous system is extended with an *oracle* or a timing assumption that circumvents that limitation of strictly asynchronous systems. We do not specify which oracle or assumption it is because instead we extend the basic system model with a multi-valued

[1] Contrary to the original Byzantine generals problem that assumed synchrony (Lamport et al., 1982).

consensus primitive, whose implementation includes the oracle/time assumption. Therefore, instead of presenting algorithms built from scratch, we present *transformations* (Hadzilacos and Toueg, 1994) from multi-valued consensus to BGAP, i.e., algorithms that convert any implementation of multi-valued consensus into BGAP. The objective is to simplify the presentation allowing us to put the emphasis on the problem itself.

2.1 Multi-valued Consensus

Suppose each process *proposes* a value, i.e., gives an initial value to the algorithm. *Multi-valued consensus* makes agreement on one of the values proposed, i.e., *decides* a value. There are a few different definitions of the problem. The definition we consider, taken from Correia et al. (2006), is given in terms of the following properties ($\perp$ is a value outside the range of values that can be proposed):

- *MVC1 Validity 1.* If all correct processes propose the same value v, then any correct process that decides, decides v.
- *MVC2 Validity 2.* If a correct process decides v, then v was proposed by some process or $v = \perp$.
- *MVC3 Validity 3.* If a value v is proposed only by faulty processes, then no correct process that decides, decides v.
- *MVC4 Agreement.* No two correct processes decide differently.
- *MVC5 Termination.* Every correct process eventually decides.

This definition has three Validity properties. Most definitions of the problem have only MVC1 or MVC2, while MVC3 is taken from the classical Byzantine generals problem (Lamport et al., 1982). Defining multi-valued consensus in terms of all these properties, makes it easier to use it for solving other problems, which is what we are interested in this paper. An algorithm that solves the problem in terms of the five properties above is given by Correia et al. (2006).

We consider that the multi-valued consensus algorithm is implemented by the function *m_v_consensus*, which takes a proposal and a consensus identifier as input parameters and returns the value decided.

2.2 Reliable Broadcast

Besides multi-valued consensus, we use a second primitive to implement BGAP: *reliable broadcast* (Bracha, 1984). This primitive ensures essentially that all correct processes deliver the same messages, which is an important restriction to what can be done by faulty processes. The problem can be defined in terms of the following properties:

- *RB1 Validity:* If a correct process broadcasts a message M, then some correct process eventually delivers M.
- *RB2 Agreement:* If a correct process delivers a message M, then all correct processes eventually deliver M.
- *RB3 Integrity:* For any identifier *ID*, every correct process *p* delivers at most one message M with identifier *ID*, and if *sender(M)* is correct then M was previously broadcast by *sender(M)* [2] .

The reliable broadcast primitive (e.g., the one by Bracha (1984)) is called using the function *r_broadcast*, which takes the message to disseminate as input.

3 Problem Definition

Each process p_i has a finite (but potentially large) set of *good values* $G_i = \{v_{i1}, ..., v_{ik_i}\}$ (good plans) and a finite (but potentially large) set of *bad values* $B_i = \{v'_{i1}, ..., v'_{il_i}\}$ (bad plans), where $v_{ij} \in \mathscr{V}$ and $v'_{ij} \in \mathscr{V}$, $\forall i, j$. Each process *decides* a value. The

[2] The predicate *sender(M)* gives the field of the message header that identifies its sender. We consider that the sender also delivers the messages it broadcasts.

BGAP problem is defined in terms of the following properties:

- *BGAP1 Validity 1.* If there is a value v such that for any correct process p_i, $v \in G_i$, then any correct process that decides, decides a value v' such that $v' \in G_j$ for a correct process p_j.
- *BGAP2 Validity 2.* No correct process p_i decides a value v if there is a correct process p_j with $v \in B_j$.
- *BGAP3 Agreement.* No two correct processes decide differently.
- *BGAP4 Termination.* Every correct process eventually decides.

Property BGAP1 is a variation of a common validity property used to define multi-valued consensus (MVC1 in Section 2.1). The property states the minimal condition in which the value decided must be a value v' of a correct process' G set: that there is at least one value v that exists in all G sets in correct processes (v can either be equal to v' or another value). The reader might expect the property to impose the value decided to be v, but this is not possible. For instance, if $\Pi = \{p_1, p_2, p_3, p_4\}$, $G_1 = G_2 = G_3 = \{50, 270\}$, $G_4 = \{270\}$, then a value in a correct process' G set will be decided as all contain 270, but the value decided can be 50 (e.g., if process p_4 is very slow). Generically, there can be a value v' (or more) in several G sets in correct processes but not in all; even existing a value v in all sets, a protocol that solves the BGAP problem may decide v' since it is usually not possible to distinguish if the value is in all or only some of the G sets in correct processes.

Property BGAP2 states that no value that exists in a correct process' B set can be decided. In fact, BGAP2 is the more innovative aspect of the BGAP problem, the statement that no "bad values" can be decided by correct processes. Properties BGAP3 and BGAP4 are the standard agreement and termination properties of consensus.

We formalize an assumption about the relation between G_i and B_i in a correct process:

- *Assumption 1:* For any correct process p_i, $G_i \bigcap B_i = \emptyset$.

Table 1 shows the four variations of the problem depending on restrictions on the relation among the processes sets G and B. Each variation adds an assumption/restriction to the problem, except variation (4) which is the problem without restrictions.

	Equal B sets	Possibly different B sets
Equal G sets	[1] $\forall i, j, (correct(i) \wedge correct(j)) \Rightarrow (G_i = G_j \wedge B_i = B_j)$	[2] $\forall i, j, (correct(i) \wedge correct(j)) \Rightarrow (G_i = G_j)$
Possibly diff. G sets	[3] $\forall i, j, (correct(i) \wedge correct(j)) \Rightarrow (B_i = B_j)$	[4]

Table 1
Four variations of the BGAP problem expressed in terms of additional assumptions

Notice that the definitions of G_i and B_i above seems to imply that G and B sets are defined by enumeration. However, this is not necessarily true, as they usually can also be defined in terms of domains and predicates (e.g., $G = \{v : v \in \mathbb{N} \wedge v < 10\}$).

4 Solutions for the problem

The four variations of the problem were ordered in terms of difficulty. We address each of the variations in this order. Each solution is a function $\mathbf{bgap}(G_i, B_i)$ that is invoked locally by each process p_i and returns the value decided v.

4.1 Variations (1) and (2)

Variation (1) has a trivial solution since all G sets are equal and all B sets are also equal. Let us define a function f which, if applied to a set of values $S = \{v_1, ..., v_m\}$ returns one of the values *deterministically*, i.e., if the function is applied to the same set S in different processes, the value returned is the same. Also, $f(\emptyset) = \perp$. Getting

momentarily back to the Byzantine generals metaphor, the function can return, e.g., the best plan in terms of expected number of casualties.

The pseudo-code for process p_i is in Algorithm 1. Variables take a subscript i to indicate that they are local variables of process p_i. The solution requires no communication.

Algorithm 1 BGAP solution for variations (1) and (2) (pseudo-code for proc. p_i).

```
1: function bgap(G_i, B_i)
2: return f(G_i)
```

This solution can be easily shown to satisfy also *variation (2)* due to Assumption 1. We skip the proofs due to the simplicity of the algorithm.

4.2 Variation (3)

Variation (3) is more interesting since it allows different processes to have different G sets. Therefore, in general, $i \neq j \Rightarrow f(G_i) \neq f(G_j)$ and the simple algorithm above does not solve the problem.

A simple algorithm that solves this variation is the following. The basic idea is to run in parallel one instance of binary consensus for each potential decision value. The algorithm executed by each process p_i is: for each possible decision value $v \in \mathscr{V} \backslash B_i$, if $v \in G_i$ then propose 1 (true), otherwise propose 0 (false) to a binary consensus with identifier v; build a vector with the decisions of all consensus executions; decide deterministically one of the values for which the decision was 1 (true).

The time complexity of this algorithm is identical to the complexity of one binary consensus, as they are all executed in parallel, which is extremely interesting. The message complexity is that of one consensus multiplied by the size of $\mathscr{V}$, so the performance of the algorithm depends on this size. However, this message complexity can be much reduced by merging and piggy-backing the messages sent in

parallel by the several consensus executions.

Algorithm 2 solves the variation (3) of the problem with a higher number of steps, but without the need of a large number of consensus instances if the size of $\mathscr{V}$ is large. The algorithm uses an operator $|S|$ that gives the number of elements in the set S. It also uses a second value $\top$ outside the range of values that can appear in G or B sets, but that can be proposed and decided by the multi-valued consensus.

Algorithm 2 BGAP solution for variation (3) (pseudo-code for process p_i).

```
 1: function bgap(G_i, B_i)
 2: w_i ← n − t
 3: r_broadcast(⟨INIT, G_i, i⟩)
 4: repeat
 5:    wait until (at least w_i INIT nessages have been delivered)
 6:    ∀_j: if (⟨INIT, G_j, j⟩ has been delivered) then VG_i[j] ← G_j else VG_i[j] ← ⊥
 7:    S_i ← {v : |{j ∈ {1,...,n} : v ∈ VG_i[j]}| ≥ t + 1}
 8:    if (S_i ≠ ∅) then r_i ← f(S_i) else r_i ← ⊤
 9:    v_i ← m_v_consensus(r_i, w_i)
10:    w_i ← w_i + 1
11:    switch (v_i)
12:       case v_i ≠ ⊤ and v_i ≠ ⊥:
13:          return(v_i)
14:       case v_i = ⊤:
15:          GG_i ← VG_i[1] ⋃ ... ⋃ VG_i[n]
16:          r_i ← f(GG_i \ B_i)
17:          v_i ← m_v_consensus(r_i, w_i)
18:          return(v_i)
19:       case v_i = ⊥:
20:          {continue the loop}
21:    end switch
22: until (w_i > n)
23: return ⊥
```

The algorithm is reasonably straightforward. Processes start by broadcasting their G sets to the others (line 3) and waiting until at least $w_i = n - t$ of these sets are delivered (line 5). Then, they build a vector with these sets (line 6) and build a set S_i with the values they received in sets from at least $t + 1$ processes (line 7). Next, the algorithm does a multi-valued consensus (line 9). The initial value (r_i) for process p_i is $f(S_i)$ if S_i is not empty, or $\top$ if it is (line 8). The outcome of the consensus is handled by the *switch* (lines 11-21). There are three cases. (1) If the

decision is neither $\top$ nor $\perp$ (line 12), then it must be one of the initial values from correct processes due to property MVC3, so it must be in a G set from a correct process and can not be in a B set from any correct process (they are all equal), so it is returned by the algorithm (line 13). (2) If the decision is $\top$ then at least one correct process (say p_k) proposed $\top$ (property MVC3), so S_k was empty, so there is no value v in all G sets from correct processes. In that case, what remains to be done is to try do do consensus on a value in some of the correct processes G sets, and not in B sets (lines 15-18). The decision of the algorithm can be either a value v that appears in some of the G sets or $\perp$ if that is not possible. (3) If the decision is $\perp$, then no agreement on a value was possible (properties MVC1, MVC2, MVC3). This is only possible if not all correct processes had the same vectors VG, meaning that they did not get the G sets from the reliable broadcast in the same order (lines 5-6). However, all processes eventually get the same messages, so it is safe to wait for one more message, i.e. for $w_i = n - t + 1$ messages (line 10), and do another consensus (line 9). The repeat loop in the algorithm does precisely that, one or more times until there are no more values to wait for.

4.2.1 Correctness

Theorem 1 *The Algorithm 2 solves variation (3) of the BGAP problem.*

Proof (sketch): The algorithm solves variation (3) of the BGAP problem if it satisfies properties BGAP1-4 given the assumption corresponding to this variation in Table 1.

BGAP1 Validity 1: Assume p_i is a correct process. If there is a value v in the condition stated in the property, then at least $n - t$ processes broadcast G sets with v (line 3), and at least $n - 2t$ of the G sets inserted in VG_i contain v (line 6). As $n - 2t \geq t + 1$, v is inserted in S_i in line 7, and $f(S_i) \neq \perp$, so a value different from $\top$ is proposed for the multi-valued consensus in line 9 (by p_i and all other correct processes). The value decided by this consensus is either $\perp$ or one of the values

proposed by correct processes (properties MVC1-MVC3), so either that valued is decided by BGAP in line 13 or the loop continues (lines 19-20). If the loop continues, exactly the same happens: no correct process proposes $\top$ and either a value different from $\perp$ is decided by BGAP or the loop continues. The loop can continue more times but the reliable broadcast delivers the same messages to all correct processes so eventually all correct processes will have the same *VG* sets, will propose the same value – which is at least in a *G* set – to the multi-valued consensus and this value will be decided by this consensus and also by BGAP.

BGAP2 Validity 2: The algorithm can only decide either a value decided by a multi-valued consensus (in lines 13 and 18) or $\perp$ (in line 23; $\perp$ can not be in a *B* set by definition). An inspection of the algorithm shows that no correct process proposes a value in its *B* set – and all *B* sets of correct processes are the same in variation (3) – and the multi-valued consensus never decides a value proposed only by faulty processes (property MVC3).

BGAP3 Agreement: An inspection of the algorithm shows that the value decided is obtained using deterministically the information returned by the multi-valued consensus executions, so all correct processes decide the same value due to property MVC4.

BGAP4 Termination: Correct processes might not decide only if they blocked: (1) in the reliable broadcast or multi-valued consensus primitives, or (2) at line 5.

Case (1). If all correct processes call *m_v_consensus* (resp. *r_broadcast*) this primitive always terminate due to property MVC5 (resp. RB1/RB2). An inspection of the algorithm shows that the sequence of lines that is executed in each correct process can vary for a single reason: the outcome of the multi-valued consensus in line 9 (the switch depends on this value). However, the value returned in line 9 is the same in all correct processes due to property MVC4, so all correct processes execute the same sequence of multi-valued consensuses and they never block (the same for the reliable broadcast, executed only once).

Case (2). A correct process might block in line 5 if it tried to wait for more *INIT* messages than would be delivered by the reliable broadcast primitive. That line never tries to wait for more than n *INIT* messages due to line 22. However, theoretically it might wait, for instance, for $n-t+1$ *INIT* messages, with t faulty processes that would not execute line 3 so only $n-t$ messages would be delivered. The proof that this situation never happens if by induction. The basis: the line never blocks when executed with $w_i = n-t$, because there are always at leat $n-t$ correct processes (at most t can be faulty). The inductive step: if the line does not block with $w_i = k$ it also does not block with $w_i = k+1$ (for $n-t \leq k \leq n-1$). The proof of the inductive step is the following. Recall that all correct processes execute the same sequence of lines. A correct process executes line 5 with $w_i = k+1$ only if it did not decide in the previous round of the loop, which implies that not all correct processes proposed the same value in line 9 (if they did, that value would be decided by the multi-valued consensus due to property MVC1, and BGAP would decide in lines 13 or 18). If not all correct processes proposed the same in line 9 with $w_i = k$, then at least $k+1$ processes must have called the reliable broadcast (line 3), or all correct processes would have the same vector *VG* (line 6) and would propose the same value for the multi-valued consensus (lines 7-9). Therefore, no correct process blocks in line 5 with $w_i = k+1$. ■

4.2.2 Efficiency

The objective of introducing the BGAP problem is not to do consensus more efficiently, but to provide an expressive abstraction that simplifies the solution of problems that involve Byzantine agreement. However, the efficiency of an algorithm is very important in practice, so this section discusses this issue.

A simple analysis of the performance of Algorithm 2 can be done by considering a few cases:

- *All correct processes have identical G sets.* The algorithm runs one reliable

broadcast and one multi-valued consensus. All correct processes propose the same value to this multi-valued consensus, something that causes the fast termination of the protocol if it is optimized for that situation (Friedman et al., 2005).

- *All correct processes have disjunct G sets.* The algorithm runs one reliable broadcast and two multi-valued consensuses (the first of which can terminate fast as all correct processes propose the same value, $\top$).
- *Other cases.* The algorithm runs one reliable broadcast and from one to t multi-valued consensuses.

This solution is efficient in 'nice' cases in which all correct processes have identical G sets or at least other favorable conditions occur like the existence of a common value that is returned by the f function in all correct processes. In other cases it runs more consensuses so it is not so efficient. A more efficient algorithm might probably be obtained by using components simpler than reliable multicast, but our purpose in this paper is to define the BGAP problem, its purpose and to provide a simple solution.

4.3 Variation (4)

Variation (4) does not impose any restriction on the relations between G sets and B sets, therefore it is the problem in its pure form. The fact that the solution for variations (1) and (2) is the same might give the idea that the same would happen for variations (3) and (4), due to the symmetry in Table 1. However, quite on the contrary: variation (4) can be shown to be unsolvable, i.e., the BGAP problem can be shown to be unsolvable without further assumptions. The intuition is that a protocol that solves the BGAP problem can not distinguish a value in a G set in a correct process from a value given by a faulty process.

Theorem 2 *The variation (4) of the BGAP problem is unsolvable.*

Proof (sketch): The proof consist in showing that there is a case in which a correct process has to choose between two options and in both cases one of the properties BGAP1/2 is broken. Consider a process p_1 with $B_1 = \{v_1\}$. Correct processes should become aware of the content of B_1 so that they do not decide v_1 if p_1 is correct. The problem is that a correct process p_2 has no way of knowing if p_1 is correct or faulty. Therefore, when p_2 somehow becomes aware of the content of B_1, it can use whatever information it has available to do one of two things:

(1) *Consider that p_1 is correct so it can not possibly decide v_1.* However, if p_1 is faulty and all correct processes have $G_i = \{v_1\}$ then property BGAP1 can not be satisfied (there is no other value in the G sets in correct processes to be decided).
(2) *Consider that p_1 is faulty so it may decide v_1.* However, if p_1 is correct and other correct processes have $G_i = \{v_1\}$ then v_1 can be decided, violating property BGAP2.

Both options may lead to the violation of the specification of the problem, so it is unsolvable. ■

What can be done about it? This impossibility result can be circumvented by making any of the assumptions of variations (1), (2) or (3). Moreover, it can be interesting to study other, possibly weaker, assumptions that allow the problem to be solved. For instance, the following assumption is sufficient to solve it:

- *Assumption 2:* For any correct process p_i and any value v, if $v \in B_i$ then there are at least $n-2t$ correct processes p_j such that $v \in B_j$.

This assumption says that a bad plan for some correct process is a bad plan for at least $n-2t$ correct processes ($t+1$ if $n = 3t+1$). An intuition of how the problem with this assumption might be solved is the following. The idea is to use a vector consensus (Correia et al., 2006) to build a set with the values at least one correct process has in its G set, and another set (BB) with the values any process has in its

B set; clearly any value that appears in a B set of a correct process has to appear in the B sets of at least $t+1$ correct processes to ensure that it is put in BB (because the vector consensus builds a vector with $n-t$ values), which is what is guaranteed by the assumption.

We believe there are other assumptions that can be used to make this variation of the problem solvable. This one has the interest of being useful to solve the third algorithm in Section 5.1.

5 Discussion

Consensus is an important problem in distributed systems, therefore many flavors have been studied through the years, even if we think only in consensus tolerant to Byzantine faults. Three important classes have to do with what is decided: binary consensus (decides 0 or 1), multi-valued consensus (decides a value $v \in \mathcal{V}$), and vector consensus (decides a vector). The BGAP problem can be considered to be closer to multi-valued consensus since it decides on a single non-binary value. However, in multi-valued consensus each process proposes a value, while in BGAP there are G sets and B sets.

Most flavors of consensus have the same Agreement and Termination properties as those we gave above for multi-valued consensus and BGAP [3] . However, for multi-valued consensus –which decides a value like BGAP– there are a few different Validity properties, usually:

- If all correct processes propose the same value v, then any correct process that decides, decides v (Dwork et al., 1988; Malkhi and Reiter, 1997; Kihlstrom et al., 2003).

[3] Exceptions are k-set consensus, which allows k different values to be decided (Chaudhuri, 1993), and randomized consensus, which terminates with probability 1 (Bracha, 1984).

- If a correct process decides v, then v was previously proposed by some process (Doudou et al., 2002; Baldoni et al., 2003).

Both validity properties have limitations. The first does not say anything about the value decided when the correct processes do not propose the same. The second lets the value decided be a value proposed by a faulty process, something that constrains the practical usage of such an algorithm. Most applications require that the value decided is proposed by a correct process, which is always true for binary consensus. For multi-valued consensus without further restrictions (like all correct processes proposing the same value), it was recently shown that the number of processes must be $(|\mathscr{V}|+1)t+1$ (Bessani et al., 2006a). However, even if the strong form of validity that can be ensured with this (high) number of processes, BGAP is more expressive.

5.1 Practical Relevance

The practical relevance of BGAP is related to these limitations of other consensus flavors. Notice that these other flavors do allow to solve many problems, including the problems that can be solved with BGAP: we implemented BGAP using a variation of multi-valued consensus. However, they are less expressive, i.e., farther than BGAP from some of the problems we want to solve.

As discussed in the introduction, BGAP takes into account the fact that processes may have several views about what decisions/actions are acceptable and unacceptable. This flexibility permits the solution of several problems. An interesting example can be taken from a recent paper that proposes cruise control algorithms to be executed by cars in a car platoon (Moniz et al., 2007). The idea is that the cars form a platoon and go together, in line, to a destination. Agreement has to be made on the speed at which the cars travel, which depends on the traffic and other environmental conditions. Agreement on the speed can be elegantly expressed in terms

of BGAP. Whenever a new speed value has to be agreed upon, BGAP is executed. Each process/car puts in the G set speed values it finds adequate, and in the B set speed values at which it can not travel (e.g., because they are too fast or the environment does not allow it). The speed values in the sets can be enumerated or defined using predicates.

The flexibility of BGAP allows also to solve the problem that was our initial motivation for studding BGAP: the implementation of Byzantine fault-tolerant tuple spaces. Recently, we proposed several algorithms to implement fault-tolerant tuple spaces using replication (Bessani et al., 2006b, 2007, 2008). In the three cases, a tuple space is replicated in a set of n servers and the system remains correct as long as no more than t servers fail. The replication algorithms we presented can be understood as solving some of the variations of the BGAP problem or, on the other hand, as being simple to implement using an algorithm that solves BGAP as a building block. The read and insert operations do not involve agreement, but the remove operation does because no tuple can be removed twice from the space (Bessani et al., 2007). Therefore, BGAP can be used to define the result of remove operations. The meaning of the sets is the following:

- G set: the set of acceptable results of the operation for the server, i.e., all tuples in the space that match the template provided;
- B set: the set of possible but unacceptable results, i.e., all tuples that match the template but were already removed from the space.

The relation of each of the algorithms with BGAP is the following:

- *Tuple space based on state machine replication:* in (Bessani et al., 2008) we presented a Byzantine fault-tolerant tuple space based on state machine replication (Schneider, 1990). The servers that implement the replicated tuple space execute the same operations in the same order so, whenever a remove operation is going to be executed, the G sets and the B sets are equal in all correct servers. This corresponds to variation (1) of BGAP (Algorithm 1).

- *Quorum-based tuple space:* variation (3) of BGAP corresponds to the quorum-based tuple space replication algorithm presented in (Bessani et al., 2006b). In this solution, a mutual exclusion algorithm is used to ensure that all removals are executed in total order in all servers so all the B sets are always equal (in correct processes). However, insertions are done using quorum algorithms, so the G sets can be different in distinct (correct) servers.
- *Linearizable tuple space:* variation (4) with Assumption 2 is used to implement tuple removal in the tuple space replication algorithm described in (Bessani et al., 2007). This algorithm requires $n \geq 4t + 1$ replicas and is based on the idea that a tuple is only removed if at least $2t + 1$ replicas report not having removed it before (so, a removed tuple cannot be reported by more than $2t$ servers).

This shows that the *Byzantine Generals with Alternative Plans problem* is quite generic, and can be used to understand several practical problems in distributed systems where Byzantine faults can happen.

Acknowledgments

We thank the anonymous reviewers for their comments that greatly assisted us in improving the paper, specially the one that suggested the first algorithm in Section 2.

This work was partially supported by the EC through project CRUTIAL and NoE ReSIST, and by the FCT through the Multiannual and the CMU-Portugal Programmes.

References

Baldoni, R., Hélary, J.-M., Raynal, M., Tangui, L., Apr. 2003. Consensus in Byzantine asynchronous systems. Journal of Discrete Algorithms 1 (2), 185–210.

Bessani, A. N., Alchieri, E., Correia, M., Fraga, J. S., Apr. 2008. DepSpace: A Byzantine fault-tolerant coordination service. In: Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems - EuroSys 2008. pp. 163–176.

Bessani, A. N., Correia, M., Fraga, J. S., Lung, L. C., Jul. 2006a. Sharing memory between Byzantine processes using policy-enforced tuple spaces. In: Proceedings of 26th IEEE International Conference on Distributed Computing Systems - ICDCS 2006.

Bessani, A. N., Correia, M., Fraga, J. S., Lung, L. C., Jul. 2007. Decoupled quorum-based Byzantine-resilient coordination in open distributed systems. In: Proceedings of 6th IEEE International Symposium on Network Computing and Applications. pp. 231–238.

Bessani, A. N., Fraga, J. S., Lung, L. C., Apr. 2006b. BTS: A Byzantine fault-tolerant tuple space. In: Proceedings of the 21st ACM Symposium on Applied Computing - SAC 2006. pp. 429–433.

Bracha, G., Aug. 1984. An asynchronous $\lfloor (n-1)/3 \rfloor$-resilient consensus protocol. In: Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing - PODC'84. pp. 154–162.

Chaudhuri, S., Jul. 1993. More choices allow more faults: Set consensus problems in totally asynchronous systems. Information and Computation 105 (1), 132–158.

Correia, M., Neves, N. F., Veríssimo, P., Jan. 2006. From consensus to atomic broadcast: Time-free Byzantine-resistant protocols without signatures. The Computer Journal 49 (1), 82–96.

Doudou, A., Garbinato, B., Guerraoui, R., May 2002. Failure detection: From crash-stop to Byzantine failures. In: International Conference on Reliable Software Technologies (Invited Paper).

Dwork, C., Lynch, N. A., Stockmeyer, L., Apr. 1988. Consensus in the presence of partial synchrony. Journal of ACM 35 (2), 288–322.

Fischer, M. J., Lynch, N. A., Paterson, M. S., Apr. 1985. Impossibility of distributed

consensus with one faulty process. Journal of the ACM 32 (2), 374–382.

Friedman, R., Mostefaoui, A., Raynal, M., Jan. 2005. Simple and efficient oracle-based consensus protocols for asynchronous Byzantine systems. IEEE Transactions on Dependable and Secure Computing 2 (1), 46–56.

Gelernter, D., Jan. 1985. Generative communication in Linda. ACM Transactions on Programing Languages and Systems 7 (1), 80–112.

Hadzilacos, V., Toueg, S., May 1994. A modular approach to the specification and implementation of fault-tolerant broadcasts. Tech. Rep. TR 94-1425, Department of Computer Science, Cornell University, New York - USA.

Kihlstrom, K. P., Moser, L. E., Melliar-Smith, P. M., Jan. 2003. Byzantine fault detectors for solving consensus. The Computer Journal 46 (1), 16–35.

Lamport, L., Shostak, R., Pease, M., Jul. 1982. The Byzantine generals problem. ACM Transactions on Programing Languages and Systems 4 (3), 382–401.

Malkhi, D., Reiter, M., Jun. 1997. Unreliable intrusion detection in distributed computations. In: Proceedings of the 10th Computer Security Foundations Workshop (CSFW97). pp. 116–124.

Malkhi, D., Reiter, M., Oct. 1998. Byzantine quorum systems. Distributed Computing 11 (4), 203–213.

Moniz, H., Neves, N. F., Correia, M., Casimiro, A., Verissimo, P., Dec. 2007. Intrusion tolerance in wireless environments: An experimental evaluation. In: Proceedings of the 13th IEEE Pacific Rim Dependable Computing Conference.

Schneider, F. B., Dec. 1990. Implementing fault-tolerant service using the state machine aproach: A tutorial. ACM Computing Surveys 22 (4), 299–319.

Vitae

Miguel Correia is Assistant Professor of the Department of Informatics, University of Lisboa Faculty of Sciences, and Adjunct Faculty of the Carnegie Mellon Information Networking Institute. He is a member of the LASIGE research unit

and the Navigators research team. He has been involved in several international and national research projects related to intrusion tolerance and security, including the MAFTIA and CRUTIAL EC-IST projects, and the ReSIST NoE. He is currently the coordinator of University of Lisboa's degree on Informatics Engineering and an instructor at the joint Carnegie Mellon University and University of Lisboa MSc in Information Technology - Information Security. His main research interests are: intrusion tolerance, security, distributed systems, distributed algorithms. More information about him is available at *http://www.di.fc.ul.pt/∼mpc*.

Alysson Neves Bessani is an Assistant Professor of the Department of Informatics, University of Lisboa Faculty of Sciences. He received his B.S. degree in Computer Science from Maringá State University, Brazil in 2001, the MSc in Electrical Engineering from Santa Catarina Federal University (UFSC), Brazil in 2002 and the PhD in Electrical Engineering from the same university in 2006. His main interest are distributed algorithms, Byzantine fault tolerance, coordination, middleware and systems architecture.

Paulo Veríssimo is currently a professor of the Department of Informatics (DI) of the University of Lisboa Faculty of Sciences (*http://www.di.fc.ul.pt/∼pjv*), and Director of LASIGE, a research laboratory of the DI (*http://lasige.di.fc.ul.pt*). He is Fellow of the IEEE. He is associate editor of the Elsevier International Journal on Critical Infrastructure Protection, and past associate editor of the IEEE Transactions on Dependable and Secure Computing. He belonged to the European Security & Dependability Advisory Board. He is past Chair of the IEEE Technical Committee on Fault Tolerant Computing and of the Steering Committee of the DSN conference, and belonged to the Executive Board of the CaberNet European Network of Excellence. He was coordinator of the CORTEX IST/FET project (*http://cortex.di.fc.ul.pt*). Paulo Veríssimo leads the Navigators research group of LASIGE, and is currently interested in: architecture, middleware and protocols for distributed, pervasive and embedded systems, in the facets of real-time adaptability and fault/intrusion tolerance. He is author of more than 130 refereed publications

in international scientific conferences and journals in the area, and co-author of five books (ex. *http://www. navigators.di.fc.ul.pt/dssa/*).