Pearls of Theory 1998:
Coordination and the Fall of the Eastern Roman Empire

Lecturer: Alessandro Panconesi

5 May 1998, week 19

Byzantium, 1453 AD. The city of Constantinople, the last remnants of the hoary Roman Empire, is under siege. Powerful Ottoman battallions are camped around the city on both sides of the Bosporus, poised to launch the next, perhaps final, attack. Sitting in their respective camps, the generals are meditating. Because of the redoubtable fortifications, no battallion by itself can succeed; the attack must be carried out by several of them together or otherwise they would be thrusted back and incur heavy losses that would infuriate the Grand Sultan. Worse, that would jeopardize the prospects of a defeated general to become Vizier. The generals can agree on a common plan of action by communicating thanks to the messenger service of the Ottoman Army which can deliver messages within an hour, certifying the identity of the sender and preserving the content of the message. Some of the generals however, are secretly conspiring against the others. Their aim is to confuse their peers so that an insufficient number of generals is deceived into attacking. The resulting defeat will enhance their own status in the eyes of the Grand Sultan. The generals start shuffling messages around, the ones trying to agree on a time to launch the offensive, the others trying to split their ranks...

Menlo Park, 1982 AD. The situation above describes a classical coordination problem in distributed computing known as *byzantine agreement* which was introduced in two seminal papers by Lamport, Pease and Shostak [1, 2]. Broadly stated, a basic problem in distributed computing is this: Can a set of concurrent processes achieve coordination in spite of the faulty behaviour of some of them? The faults to be tolerated can be of various kinds. The most stringent requirement for a fault-tolerant protocol is to be resilient to so-called byzantine failures: A process incurring a byzantine failure can behave in any arbitrary way, even share information and conspire together with other faulty processes in an attempt to make the protocol work incorrectly. Obviously, the identity of faulty processes is unknown.

What kind of applications need this kind of malicious fault to be handled? Part of the answer can be found in the footnote of the original paper by Lamport, Shostack and Pease listing the sponsors: NASA, the Ballistic Missile Defense Systems Command and the Army Research Office. Besides the usual applications to killing and terrorizing people, protocols that withstand byzantine failures are useful in any situation where it is vital that a system perfoms correctly in spite of the malfunctioning of some its subparts, regardless of the type of mulfunctioning. Examples include aircraft control systems and applications of computer technology to medicine. According to Nancy Lynch's book "the agreement problem is a simplified version of a problem that originally arose in the development of on-board aircraft control systems. In this problem, a collection of processors, each with access to a separate altimeter, and some of which may be faulty, attempt to agree on the airplane's altitude. Byzantine agreement algorithms have also been incorporated into the hardware of fault-tolerant multiprocessor systems; there, they are used to help a small collection of processors to carry out identical computations, agreeing on the results at every step. This redundancy allows the processors to tolerate the (Byzantine) failure of one processor. Byzantine agreement algorithms are also useful in processor fault diagnosis, where they can permit a collection of processors to agree on which of their number have failed (and should therefore be replaced or ignored)" [3].

The problem we want to study is whether coordination among processes is at all possible in spite of byzantine faults. We begin with a precise formulation of the problem.

We are given a set G of processes, called *generals*, which is partitioned into two sets L and T of, respectively, *loyal generals* and *traitors*. Henceforth, t denotes the number of traitors and n the overall number of processes. Each general G_i has an input bit b_i– G_i's initial assessment– and must produce an output bit d_i called the *decision value* of G_i. The underlying communication mechanism is

- **Synchronous:** The generals have perfectly synchronized clocks and a message is guaranteed to be delivered in one time unit;

- **Reliable:** Messages can neither be forged nor corrupted nor lost;

- **Authenticated:** The identity of the sender is known to the reciever; and

- **Point-to-point:** The underlying topology is that of a complete graph.

As we shall see, each of these assumptions has a big impact on the problem. The question we ask is whether there exists a protocol satisfying the following conditions.

> **Non-triviality**: If all generals have the same input bit b then, the only possible decision value of the loyal generals is b. More formally, $\forall G_i, b_i = b \Rightarrow \forall G_j \in L, d_j = b$.
>
> **Agreement**: The loyal generals should agree on the decision. That is, $\forall G_i, G_j \in L,\ d_i = d_j$.
>
> **Limited bureaucracy**: The protocol must terminate.

The first requirement is a technical condition to disallow trivial constant solutions; if all generals have the same input 0 (resp. 1), then all loyal generals must decide 0 (resp. 1). The output condition refers to loyal generals only because we have no control over the behaviour of faulty processors. The second condition captures the essence of the problem, while the third is an obvious requirement that must be satisfied for the protocol to be of any use. Note however a subtle point. The requirement can be relaxed as follows:

> **Limited dithering**: Eventually all loyal generals should come to a decision.

This is less stringent than Limited-bureacracy because a process might commit to a decision value and yet keep participating to the protocol which might even run forever. There are protocols which satisfy Limited-dithering but not Limited-bureacracy.

Definition 1. *A protocol satisfying Non-triviality, Agreement and Limited-Bureacracy is called a* consensus protocol *or* agreement protocol. *The* byzantine agreement *problem is that of finding a consensus protocol that withstands byzantine failures.*

Definition 2. *A protocol is t-*resilient *if it tolerates up to t failures.*

With this terminology, whe can state our original problem more precisely: What is the maximum t for which a t-resilient protocol for byzantine agreement exists? The answer is given by the following theorem.

Theorem 1. *Let the underlying network topology be that of a complete graph. Then, there exists a t-resilient protocol for byzantine agreement among n processes if and only if $t < \frac{n}{3}$.*

Our aim in this note is to prove the only-if part: If $t \geq \frac{n}{3}$ then byzantine agreement is impossible. Before proving the result, let's return to the problem of the impact of the various hypothesis made. A basic insight gained by the research in distributed computing is that there is a huge difference in power between synchronous and asynchronous systems. A *crash failure* of a process p occurs when p stops functioning forever (p is "dead"). This type of fault is obviously much easier to handle than byzantine failures. The difference between synchrony and asynchrony is nicely exemplified by the following celebrated impossibility result.

Theorem 2. *There is no protocol for agreement in asynchronus systems that withstands 1 (or more) crash failures.*

In other words, if the system is synchronous, it is possible to tolerate the *byzantine failure* of up to one third of the processes, whereas if the system is asynchronous not even one crash failure can be tolerated.

The connectivity of the underlying network too has a great impact on the feasibility of the problem. The precise characterization of the network topology required to solve byzantine agrement is based on a basic graph theory result known as Menger's Connectivity Theorem.

Theorem 3. *A t-resilient protocol for byzantine agreement in synchronous systems exists if and only if (a) $t < \frac{n}{3}$ and (b) the network is $(2t+1)$–vertex connected, i.e. between each pair of distinct vertices there are at least $2t+1$ mutually vertex disjoint paths.*

Another fundamental insight into computation is given by a set of results that show that randomized algorithms are more powerful than deterministic ones. A randomized protocol is an algorithm in which processors can flip coins (i.e. invoke a random number generator and decide accordingly) Consider for instance this theorem.

Theorem 4. *There exists a randomized protocol for agreement in asynchronous systems that withstands up to $t < \frac{n}{2}$ crash failures. The protocol reaches agreement with probability 1.*

That is, whereas deterministic protocols cannot even tolerate 1 crash failures, randomized protocols can withstand the crash failure of up to half of the processes. In fact, more recent research has considerably strengthened the above result which was achieved by making use of a rather time-ineffiecient protocol. Recent randomized protocols are very efficient and can tolerate up to $n-1$ crash failures!

Let's now turn to our impossibility proof for byzantine agreement.

The Impossibility Proof. Let's state the result we want to prove once again: There exists no t–resilient protocol for agreement that can withstand $t \geq \frac{n}{3}$ byzantine failures. Recall that we are considering synchronous systems.

We start by introducing a model for the system. Processes are deterministic automata with a state space S which is possibly infinitely large (this will make our impossibility result stronger). Each process has its own ID. Without loss of generality let i be the ID of p_i– the i-th process. The input of p_i, the initial assesment, is b_i and its output is d_i. Let $in_k(i)$ and $out_k(i)$ denote the set of messages sent and received by p_i at round k. The states of p_i and the messages sent by it are determined by three deterministic functions as follows:

$$\begin{aligned} s_0(i) &:= F(i, b_i) \\ s_{k+1}(i) &:= G(i, s_k(i), in_k(i)) \\ out_{k+1}(i) &:= H(i, s_k(i), in_k(i)) \end{aligned} \tag{1}$$

In other words, a *protocol* consists of 3-tuple of functions (F, G, H). The processes undergo the above state transitions simultaneously at round k. This reflects (models) the synchronous nature of the system. The deterministic nature of the protocol is an essential ingredient of our impossibility proof. We first settle the case $n = 3$. Perhaps surpisingly, this is the difficult step.

Theorem 5. *There is no 1-resilient protocol for byzantine agreement when $n = 3$.*

Proof. We assume that a 1-resilient protocol exists and derive a contradiction. Let's begin with an outline of the proof. We will test the alleged

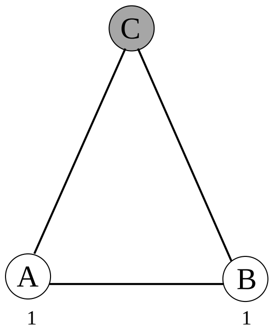

Figure 1: The first scenario

protocol on different situations and show that it cannot possibly be correct in all cases. First, consider the scenario in Figure 1. In this scenario we give A and B input 1 and make C faulty. By Non-triviality and Limited-Beauracracy both A and B are forced to decide 1. The exact behaviour of the faulty process C will be specified later.

Second, we consider the scenario in Figure 2. Here, B and C are given input 0 and A is made faulty. Again, by Non-triviality and Limited-bureaucracy the protocol must decide 0 for both B and C. Third, we consider the scenario in Figure 3. Here, we give A input 1, C input 0, and we make B faulty. The crucial point is that the behaviour of the faulty processes in the three scenarios can be specified in such a way that A will receive the same sequence of messages as in the first scenario and therefore, since its input is the same in both scenarios, it will execute the same set of instructions and output 1 (the protocol is deterministic!). Similarly, C will receive exactly the same

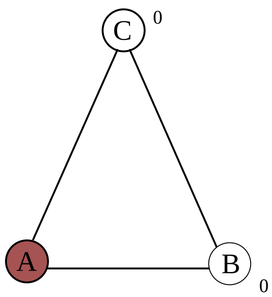

Figure 2: The second scenario

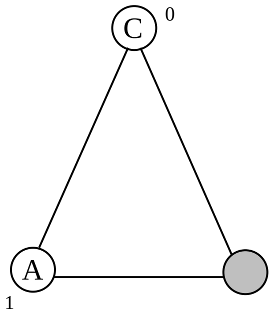

Figure 3: The third scenario

messages as in the second scenario and, having the same input as in that situation, it will decide 0. But this violates Agreement, implying that the protocol cannot be correct.

That's the plan. The problem is how to specify the behaviour of the faulty processes in the three scenarios. To this aim, we take three processes A,B and C and set up a physical system as indicated in Figure 4. Two identical copies of each process are placed at the opposite corners of the hexagon. For notational convenience we use primed letters to distinguish the two copies of each process. It should be born in mind that $a := id(A) = id(A')$, $b := id(B) = id(B')$ and $c := id(C) = id(C')$, and that the same deterministic protocol (F, G, H) is executed at all sites. What changes from one site to the next is the input bit or the process' ID. Twin processes receive different inputs as specified in figure. In the hexagon the protocol is not supposed to satisfy any particular requirement. Nevertheless, it will exhibit a specific behaviour because once the ID's and the input bits are specified the sequence of states entered by and of messages sent by the processes is completely determined. By looking closely at this behaviour we will be able to find out how to specify the behaviour in the three scenarios so that the desired contradiction is reached. To start with, the initial states are completely determined and are as follows:

$$\begin{aligned} s_0(A) &= F(a,1) \\ s_0(A') &= F(a,0) \\ s_0(B) &= F(b,1) \\ s_0(B') &= F(b,0) \\ s_0(C) &= F(c,1) \\ s_0(C') &= F(c,0) \end{aligned}$$

Similarly, the first set of outgoing messages is completely determined:

$$\begin{aligned} out_1(A) &= H(a, s_0(A), \text{NIL}) \\ out_1(A') &= H(a, s_0(A'), \text{NIL}) \\ out_1(B) &= H(b, s_0(B), \text{NIL}) \\ out_1(B') &= H(b, s_0(B'), \text{NIL}) \\ out_1(C) &= H(c, s_0(C), \text{NIL}) \\ out_1(C') &= H(c, s_0(C), \text{NIL}) \end{aligned}$$

This means, of course, that the first set of incoming messages is also determined. In turn, this specifies the set of states at round 1, and so on round after round. The behaviour of the system is completely specified.

We are now ready to specify the behaviour of the faulty processes in the three scenarios. We shall denote by m_i^{XY} the message sent by X to Y at round i in the hexagon. Refer to Figure 5, showing the three scenarios again, and focus on the first, leftmost scenario. Intuitively, we want to specify C's behaviour so that it behaves as C with respect to B in the hexagon, and as C' with respect to A in the hexagon. Specifically, we specify C's outgoing messages as follows: at round i, C will send $m_i^{C'A}$ to A and m_i^{CB} to B. Notice that this implies that A and B will behave exactly as they do in the hexagon; they will go through the same sequence of states and their outgoing messages will be $m_i^{AC'}$ and m_i^{AB} for A, and m_i^{BA} and m_i^{BC} for B (this follows from an easy induction). As remarked, the alleged correctness of the protocol implies that $d_A = d_B = 1$.

We now switch to the second scenario of Figure 5. Here, B' and C' have input 0 and A is made faulty– it will behave as A in the hexagon with respect to C' and as A' with respect to B'. More precisely, we specify A's outgoing messages as follows: the messages to B' will be $m_i^{A'B'}$ while those to C' will

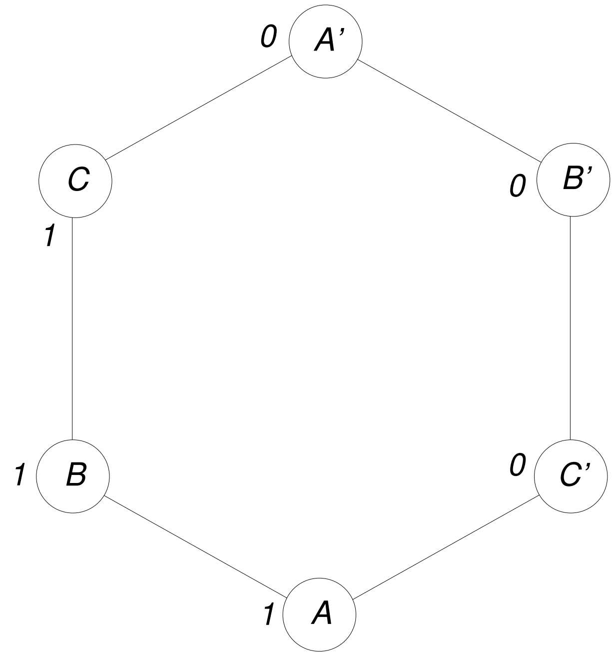

Figure 4: The hexagon of generals.

be $m_i^{AC'}$. Again, B' and C' will go through the same sequence of states as in the hexagon and their outgoing messages will be $m_i^{C'B'}$, $m_i^{C'A}$, $m_i^{B'C'}$, and $m_i^{B'A'}$. Since B' and C' have both input 0 and the protocol is supposed to be 1-resilient, they must agree on 0, i.e. $d_{B'} = d_{C'} = 0$.

In the final scenario we reach the desired contradiction, showing that the alleged correct protocol violates agreement. Here we give A input 1, as in the first scenario, we give C' input 0, as in the second scenario, and make the third process faulty by specifying its outgoing messages as follows: at round i, m_i^{BA} is sent to A and $m_i^{B'C'}$ is sent to C'. As before, the intuition is that the third process is behaving as B in the hexagon with respect to A and as B' in the hexagon with respect to C'. Notice now that A has the same input bit as in the first scenario and that *it receives exactly the same sequence of incoming messages as in that scenario*. Therefore it will execute the same sequence of states and output 1. Analogously, C' will behave as in the second scenario and output 0. But this violates Agreement. Hence, the protocol cannot be 1-resilient. $\square$

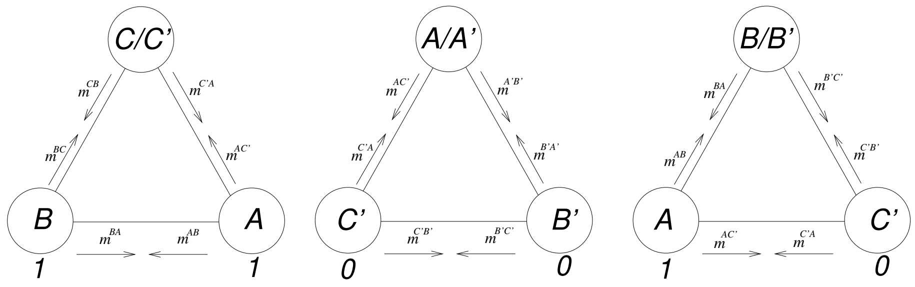

Figure 5: The three scenarios.

This result can now be extended for $n > 3$.

Corollary 1. *Let $n \geq 1$. Then there is no t-resilient protocol for the Byzantine generals for $t \geq n/3$.*

Proof. Homework. $\square$

Comments. The fall of Constantinople is recounted by Stefan Zweig in his novel "Die Eroberung von Byzanz" one of the short stories in the collection "Sternstunden der Menscheit". In truth, the term "byzantine agreement" is a misnoumer— "ottoman agreement" would be more appropraite. Of course, one could always change the story...

References

[1] L. Lamport, R. Shostak, and M. Pease, The byzantine generals problem, ACM Trans on Prog Lang and Syst, Vol. 4, No. 3, July 1982, Pages 382–401

[2] M. Pease, R. Shostak, and L. Lamport, Reaching agreement in the presence of faults, JACM, Vo. 27, No.2, April 1980, pp. 228–234

[3] N. A. Lynch, Distributed Algorithms, Morgan Kaufmann Publishers, Inc., San Francisco 1996

[4] B. Chor and C. Dwork, Randomization in Byzantine agreement. In S. Micali, editor, Randomness and Computing (Advances in Computing Research, vol. 5). pages 443–497. JAI Press, Greenwich, CT, 1989.