

Interconnecting Lock-step Synchronous Fault-tolerant Systems based on Voting and Error-Correcting Codes

Thijs Krol

University of Twente, Department of Computer Science
P.O. Box 217, NL7500 AE Enschede, The Netherlands
email: krol@cs.utwente.nl

Abstract

The correctness of the behavior of a fault-tolerant system depends among other things on the correct distribution of the data descending from unreliable I/O devices over the modules of the fault-tolerant system, the so-called input-problem. More generally, a maliciously behaving system, whether it is fault-tolerant or not, should never defeat a correctly functioning fault-tolerant system, i.e. a system which does not contain more faulty modules than it is designed to tolerate.

This paper presents a new class of synchronous deterministic non-authenticated algorithms for reaching Byzantine agreement on data descending from other (fault-tolerant) devices. The algorithms are based on voting and error-correcting codes and require considerably less data communication than the existing algorithms, whereas the number of rounds and the number of modules meet the minimum bounds.

1 Introduction

Fault-tolerant system (ft-system) in practice are always built from a number of different fault-tolerant subsystems, each having different reliability requirements and properties. Such a system could, for example, consist of subsystems that are divided into three distinct reliability levels, viz. the external inputs that are basically unreliable, a reliable reconfigurable redundant storage system and a supervisory service with high reliability. In any case a ft-system will be built from a fault-tolerant part and unreliable input-devices. Communication between these subsystems must be such that data originating from a malfunctioning subsystem never causes the receiving system to break down. So the interconnection between these ft-subsystems has to be done very carefully.

Without applying special precautions, broadcast faults (Byzantine fault) in which a malfunctioning source sends different data to the various modules of a receiving ft-system, might cause a correctly function-

ing ft-receiving system to go down, even if the number of faults in the receiving system is not more than it is designed to tolerate.

In any ft-system the data are replicated, or encoded with some error-correcting code [5], and distributed over a number of modules such that the data can be recovered in case one or more modules fail. Similarly, the processes (functions) which act on this data are replicated and distributed over a number of modules such that the data that result from these processes is again fault resilient. So in a ft-system the data remains recoverable (fault resilient) as long as the number of faulty modules does not exceed a certain number. However this fault resilience depends on the fact that the input data for the replicated processes is fault resilient too. So all correct processes must start from the same data, which is not guaranteed in case of broadcast (Byzantine) faults.

Broadcast faults in the communication between a transmitting and a receiving ft-system, may be caused by power, driver and clock failures. The physical processes behind these failures are slow compared to the data transmission rate and therefore ambiguous data values, which are interpreted differently by the receiving modules, are likely to be sending by the transmitting system.

One should distinguish between the probably incorrect data that is produced by a malfunctioning source and the fact that a malfunctioning source sends different data to the various modules of a fault tolerant system. The application programs should be able to keep up with incorrect data while the communication protocols between the ft-systems must cope with the broadcast faults. The latter will be called the *input-problem*. The algorithms that solve the input-problem will be called *input-agreement algorithms*.

The input-agreement algorithms are related to, but different from the consensus and interactive consistency algorithms.

With consensus a set of modules is considered, of which at most T are malfunctioning. Each module possesses an initial value v_i . After completion of the protocol, the correct receiving modules all decide on the same value and if all initial values are equal, say v , then all processors decide on the value v .

With interactive consistency again a set of modules is considered each possessing an initial value. In this case, after completion of the protocol, all modules agree on the initial value they think they have received from each of the modules. For every module p it holds that if p is correct, the above mentioned agreement equals the initial value actually sent by processor p .

An interactive consistency (IAC) algorithm, in which only one source module is considered, is also called a Byzantine agreement algorithm. For detailed classification of consensus and IAC algorithms we refer to [10], chapter 3 and [1].

Opposite to the consensus and IAC algorithms, in which only a single ft-system is considered, the input-agreement algorithms deal with a system consisting of two distinct subsystem, a possibly ft-transmitting system, the t-system, which may be malfunctioning, and a ft-receiving system, the r-system. Notice that a ft-system is functioning correctly if it contains no more faults than it is designed to tolerate. Input-agreement requires that all correctly functioning modules in the correctly functioning r-system agree on the value they receive from the t-system and if the t-system is functioning correctly this value corresponds to the value which was actually sent by the t-system.

The input-problem defined in this way has been addressed for first time in [4, 5]. In the current paper a new class of input-agreement algorithms is described which includes the ones published in [4], but additionally contains algorithms which require considerably less communication. These algorithms are based on symbol-error-correcting codes and take advantage of the IAC algorithms described in [7]. The algorithms described in this paper require a lock-step synchronous environment and are non-authenticated.

In section 2 we will give a precise description of our system model and a definition of the input-agreement problem. In section 3 a few results of [7] needed in this paper, will be summarized. Thereafter in section 4 and 5 two classes of input-agreement algorithms will be described and its properties will be proven. In section 6 the various algorithms will be compared and discussed.

2 The system model

We consider a ft-system consisting of two subsystems, the t-system and the r-system. The t-system trans-

mits data to the r-system. The t-system consists of a set \mathbf{Ns}_t of n_t modules ($|\mathbf{Ns}_t| = n_t$) and tolerates up to and including T_t faulty modules. Similarly the r-system is characterized by \mathbf{Ns}_r , n_r and T_r . In case we are dealing with a single non-fault-tolerant t-system $n_t = 1$ and $T_t = 0$. We consider only lock-step synchronous non-authenticated systems, so fault-tolerance is implemented by masking. Therefore, the n_t output values (symbols) of the t-system form a code word of a symbol-error-correcting code consisting of n_t symbols that allows the correction of T_t erroneous symbols at arbitrary locations, [5, 7]. If the t-system is an N-modular redundant (NMR) system, this symbol-error-correcting code boils down to a simple repetition code consisting of n_t identical symbols. In case of a single non-redundant t-system the encoding function is just a simple identity function.

Let x represent the original data value available in the t-system that has to be transmitted to the r-system. Then each of the n_t modules in the r-system possesses one symbol of the encoded version of x . These n_t symbols are denoted as a function m_t on the modules of the t-system. So $m_t(i_t)$ with $i_t \in \mathbf{Ns}_t$ is the symbol possessed by module i_t of the t-system. The encoding function is modeled by \mathcal{T} such that $m(i_t) = \mathcal{T}(i_t)(x)$ is the symbol of the encoded version of x which is possessed module i_t .

Starting from the model described above, we define *input-agreement* as follows:

Definition 1 *Input-agreement is satisfied if:*

- *In a correctly functioning r-system, i.e. at most T_r modules are malfunctioning, the well-functioning modules d_r , e_r always agree with each other on the decoded messages they calculate from the data they received from the t-system, i.e. $dec(d_r) = dec(e_r)$.*
- *If the t-system is functioning correctly, i.e. at most T_t modules are malfunctioning, the above-mentioned agreement should equal the decoded value x of the data actually sent by the t-system, i.e. $dec(d_r) = x$. □*

Without proof we state that input-agreement in a lock-step synchronous non-authenticated environment can only be satisfied if $n_r \geq 3T_r + 1$.

3 Interactive consistency algorithms which are based on voting and error-correcting codes

IAC algorithms are defined on a set \mathbf{Ns} of n fully interconnected modules. Provided the number of malfunctioning modules is at most T , all correct processors

decide on the same result and if the source a is correct, this result equals the data actually sent by the source. It has been shown that non-authenticated lock-step synchronous algorithms only can exist if $n \geq 3T + 1$ [9] and require at least $T + 1$ rounds of communication [3].

The input-agreement algorithms described in this paper and the way in which they are constructed heavily rely on the IAC algorithms described in [7]. These non-authenticated lock step synchronous IAC algorithms are based on the application of symbol-error-correcting codes which considerably reduces the communication between the modules.

Because many IAC algorithms exist having the same properties, classes $\mathcal{A}(T, T+1, a, \mathbf{Ns})$ of IAC algorithms have been defined. These classes contain algorithms which tolerate T faulty modules and transmit in $T+1$ communication rounds an initial message from the source a to all modules in the set \mathbf{Ns} . Notice that $a \in \mathbf{Ns}$ and $|\mathbf{Ns}| \geq 3T + 1$. A particular algorithm in a class $\mathcal{A}(T, T + 1, a, \mathbf{Ns})$ lays down the algorithm in detail.

The T -symbol-error-correcting codes used in the algorithms are characterized by the parameters (n, k, b) , in which n is the number of symbols in a code word, k is the number of data symbols and b is the number of bits per symbol. These codes do exist if $k \geq 1$, $n - k \geq 2T$, $b \geq 1$ and $k \geq 2 \implies b \geq \log_2(n - 1)$, [8]. For efficiency reasons we will always choose $n - k = 2T$, see [7].

In an algorithm from the class $\mathcal{A}(T, T + 1, a, \mathbf{Ns})$ the original message in the source module a is encoded with a (n_1, k_1, b_1) code and each of the n_1 symbols is sent to a different module, not being the source module. The receiving module treats the received message in the same way as a source module does and thus encodes the received message (symbol) again, with a code (n_2, k_2, b_2) , and forwards the resulting symbols again to different modules that were not yet visited. In round $T + 1$ the received symbols are sent unchanged to all modules which did not yet receive the message.

After these $T + 1$ communication rounds, in each module a decision-making process of $T + 1$ decision rounds is executed. During the decision-making process no communication between the modules takes place. Successively in each module the decoder functions (n_T, k_T, b_T) to (n_1, k_1, b_1) are applied on the received and calculated data, which finally yields an estimate of the initial message in the source. A particular algorithm thus is characterized by the sequence of coding and decoding functions applied by the algorithm. Each of them is denoted by the applied code. For

example the most simple algorithm with $T = 1$ and $N = 4$ is characterized by $[3, 1, 1]_1 t f 2_3 \langle 3, 1, 1 \rangle_3$, in which $[n, k, b]$ and $\langle n, k, b \rangle$ mean encoding and decoding with a (n, k, b) code respectively, t means transmission and $f2$ means forwarding each symbol to 2 modules. The number of times an encoding, decoding or forwarding is applied in the system is added in subscript.

A detailed discussion on the construction of these algorithms can be found in [6, 7].

4 Input-agreement algorithms based on post-observation

Let a fault-tolerant system consist of two subsystems as defined in section 2, the t-system which might be fault-tolerant and the r-system which is fault tolerant. Fault-tolerance is implemented by means of masking redundancy. The t-system transmits data to the r-system. We do not want such a t-system to cause the receiving t-system to go down, regardless of whether the t-system is functioning correctly or behaving maliciously.

The data in the t-system is available in the form of an T_t error-correcting code word, such that each module possesses one symbol of the code word. The basic idea, behind this input-agreement algorithm is that each symbol of this code word is encoded with an T_r error-correcting code, after which each symbol of the code word this yields is sent to a different module of the r-system, the input-modules. These input modules relay the received data to all modules of the r-system by means of an IAC algorithm. Hence in all modules of the r-system code words become available, which are the encoded versions of the symbols in the t-system. On these code words the decoder function of the T_r error-correcting code is applied. After that the code word of the T_t error-correcting code might be decoded. Agreement is obtained from the IAC algorithm and the influence of faulty input modules is masked by the T_r error-correcting code.

The preceding construction boils down to the following definition

Definition 2 (Post observation) *Let the observation function of the t-system be denoted by $\mathcal{T}^{(-1)}$, then*

- *In all modules i_t of the t-system the messages $m(i_t)$, $m(i_t) = \mathcal{T}(i_t)(x)$, are encoded by means of a T_r -error-correcting code \mathcal{W}_{i_t} . Each of the resulting symbols of a code word is sent to a different module a_r of the r-system, $a_r \in \mathbf{Inp}(i_t)$. A module a_r forwards this symbol to all modules of the r-system by means of an interactive consistency algorithm of the class $\mathcal{A}(T_r, T_r + 1, a_r, \mathbf{Ns}_r)$.*

- In each module d_r the interactive consistency algorithms bring forth decisions $dec((i_t, a_r)d_r)$ about the values $m(i_t, a_r)$ received by the modules a_r from i_t . On these decisions for each i_t , first the decoder function $\mathcal{W}_{i_t}^{-1}$ is applied, resulting in decisions $dec((i_t)d_r)$ and thereafter on the latter decisions, again in each module d_r , the observation function $\mathcal{T}^{(-1)}$ is applied.

The indication “post-observation” stems from the fact that the observation function \mathcal{T} is applied after the IAC algorithms are applied.

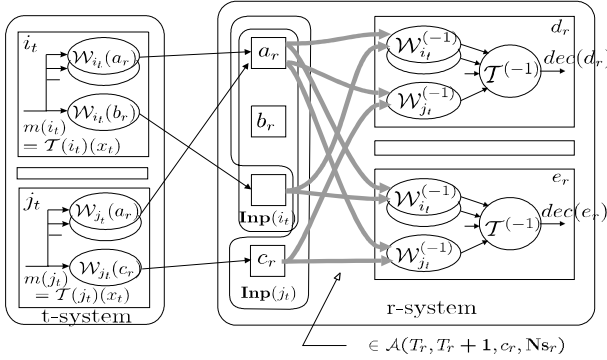


Figure 1: **The algorithm based on post-observation**

The algorithm is shown in figure 1. In this figure the ft-subsystems and the sets $\mathbf{Inp}(i_t)$ are indicated by boxes with rounded corners, modules by square boxes, direct communications by arrows and function applications by ovals. The communications due to the algorithms from the classes $\mathcal{A}(T_r, T_r + 1, a_r, \mathbf{Ns}_r)$ are denoted by \longrightarrow .

Theorem 1 *The input-agreement algorithm based on post-observation defined in definition 2 satisfies the input-agreement properties defined in definition 1. \square*

Proof:

Let the r-system be functioning correctly, i.e. the number of malfunctioning modules is at most T_r . The messages $m(i_t, a_r)$ received by the r-system from the t-system with $i_t \in \mathbf{Ns}_t$ and $a_r \in \mathbf{Inp}(i_t)$, are all forwarded to all modules d_r of the r-system by means of an interactive consistency algorithm. Each of these algorithms results in each module d_r in a decision $dec((i_t, a_r)d_r)$ about the value $m(i_t, a_r)$. Because the r-system is functioning correctly we obtain from the properties of interactive consistency that for all i_t , a_r and correct d_r and e_r holds $dec((i_t, a_r)d_r) = dec((i_t, a_r)e_r)$ (agreement). Moreover, because a_r is the source module in the IAC algorithm, we know that if a_r is correct $dec((i_t, a_r)d_r) = m(i_t, a_r)$.

Let d_r and e_r be two correctly functioning in the r-system. The messages $m(i_t, a_r)$ received by the modules a_r , with $a_r \in \mathbf{Inp}(i_t)$, from the modules i_t are the symbols of the encoded message $m(i_t)$. Hence, if i_t and a_r are correct then $m(i_t, a_r) = \mathcal{W}_{i_t}(a_r)(m(i_t))$. In the decision making process in each module d_r , for each i_t , the decoding function $\mathcal{W}_{i_t}^{(-1)}$ is applied on the decisions $dec((i_t, a_r)d_r)$ with $a_r \in \mathbf{Inp}(i_t)$, resulting in decisions $dec((i_t)d_r)$. We already concluded that if a_r is correct, $dec((i_t, a_r)d_r) = m(i_t, a_r)$. So in case module i_t is correct and because at most T_r modules a_r in $\mathbf{Inp}(i_t)$ are faulty, we obtain $dec((i_t)d_r) = m(i_t)$. We know that $m(i_t) = \mathcal{T}(i_t)(x)$, provided i_t is functioning correctly and the t-system contains at most T_t faulty modules. During the last step of the decision making process, in each module d_r of the r-system, the observation function $\mathcal{T}^{(-1)}$ of the t-system is applied on the decisions $dec((i_t)d_r)$ with $i_t \in \mathbf{Ns}_t$, bringing forth decisions $dec(d_r)$. So $dec(d_r) = x$ if at most T_t modules of the t-system are malfunctioning. (property 2 of definition 1).

Recall that $dec((i_t, a_r)d_r) = dec((i_t, a_r)e_r)$ for any two correct modules d_r and e_r . In all modules the same functions $\mathcal{W}_{i_t}^{(-1)}$ and $\mathcal{T}^{(-1)}$ are applied. Hence for the results $dec(d_r)$ and $dec(e_r)$ this yield must hold $dec(d_r) = dec(e_r)$, (property 1 of definition 1)

This completes the proof of Theorem 1. \square

5 Input-agreement algorithms based on pre-observation

5.1 Introduction

The pre-observation method is an efficient alternative to the post-observation method, i.e. it requires less data communication on the account of a little more complex coding scheme.

In the algorithm based on post-observation, The observation function $\mathcal{T}^{(-1)}$ is applied after the IAC algorithms has been applied, (post-observation). By combining the two encoding functions \mathcal{T} and \mathcal{W} into a *product code* we are able to change the order in which the decoding functions $\mathcal{W}^{(-1)}$ and $\mathcal{T}^{(-1)}$ are applied, in such a way that the observation function $\mathcal{T}^{(-1)}$ is applied immediately after the data has been received by the r-system. That is before the IAC algorithms are applied, (pre-observation).

5.2 Product codes

A product code [8] with parameters $(n_t.n_w, k_t.k_w, b)$ is built from two linear codes with parameters (n_t, k_t, b) and (n_w, k_w, b) respectively. The product code is constructed by arranging the $n_t.n_w$ symbols of a word of the product code in an $n_t \times n_w$ matrix such that the columns of the matrix are code words of the (n_t, k_t, b)

code and the rows of the matrix are code words of the (n_w, k_w, b) code. The Hamming distance of a product code is the product of the Hamming distances of the underlying codes. This property, however, will not be used. Instead we will use in our input-agreement protocol two other well-known properties of product codes. These properties are expressed in the lemmas 1 and 2 respectively. The proofs can be found in almost any textbook on error correcting codes.

Lemma 1 *A code word of an (n_t, n_w, k_t, k_w, b) product code can be constructed from a matrix of $k_t \times k_w$ data symbols by either first encoding the k_w columns with the (n_t, k_t, b) code and thereafter encoding the n_t rows, this yields, with the (n_w, k_w, b) code, or alternatively, by first encoding the rows with the (n_w, k_w, b) code and thereafter encoding the resulting n_w columns with the (n_t, k_t, b) code.*

As a consequence, decoding can be applied in any order too. \square

Lemma 2 *An (n_t, n_w, k_t, k_w, b) product code that is constructed from a T_t -symbol-error-correcting code with parameters (n_t, k_t, b) and a T_r -symbol-error-correcting code with parameters (n_w, k_w, b) allows the simultaneous correction of T_t rows and T_r columns of random symbol errors.* \square

In figure 2 the encoding and decoding possibilities of a $T \times W$ product code are elucidated. In which \mathcal{T} and \mathcal{W} are an (n_t, k_t, b) code and an (n_w, k_w, b) code respectively.

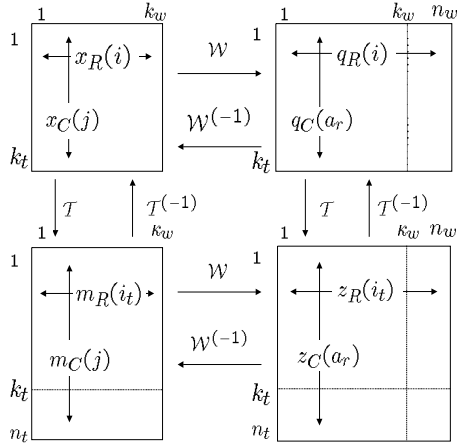


Figure 2: **Encoding and decoding sequences in a product code**

5.3 The algorithm

Again we start from the system model described in section 2, i.e. t-system which transmits data to the

fault-tolerant r-system. In both the t-system and the r-system, fault-tolerance is obtained by masking redundancy, such that the data in the t-system can be retrieved with an observation function $\mathcal{T}^{(-1)}$. In this case the initial value x consists of a matrix of $k_t \cdot k_w$ symbols. The columns of this matrix are encoded by the t-system with the T_t -error-correcting code \mathcal{T} . The algorithm encodes the n_w rows with a T_r -error-correcting code \mathcal{W} . The word z (an $n_t \times n_w$ matrix) of the product code $\mathcal{T} \times \mathcal{W}$, this yields, is sent to a set **Inp** of modules of the r-system, such that a module $a_r, a_r \in \mathbf{Inp}$ receives a column of z , i.e. a code word of \mathcal{T} . Each module a_r in **Inp** decodes this code word and forwards the result to all modules of the r-system by means of an IAC algorithm. On these results the decoding function $\mathcal{W}^{(-1)}$ is applied.

This leads to the following definition and theorem:

Definition 3 *Let the initial data value x in the t-system consist of $k_t \cdot k_w$ symbols, i.e. a matrix consisting of k_w columns $x_C(j)$ of k_t symbols and let the data in the t-system be encoded with a T_t -error-correcting code \mathcal{T} , such that each module i_t possess k_w symbols $m(i_t, j)$ with $m(i_t, j) = \mathcal{T}(i_t)(x_C(j))$. Furthermore, let **Inp** be a set of n_w modules in the r-system with $2T_r + 1 \leq n_w \leq n_r$ which provides for the communication from the t-system.*

- Each module i_t in the t-system encodes the word (row) consisting of k_w symbols $m(i_t, j)$ with a T_r -error-correcting code \mathcal{W} . The n_w symbols $z(i_t, a_r)$ with $a_r \in \mathbf{Inp}$, this yields, are each transmitted from module i_t to module a_r .
- The modules a_r in **Inp** each have received n_t symbols and each apply on the received word of n_t symbols the observation function of the T -system $\mathcal{T}^{(-1)}$. Thereafter, each module a_r forwards the result of the decoding to all modules of the r-system by means of IAC algorithms of the class $\mathcal{A}(T_r, T_r + 1, a_r, \mathbf{Ns}_r)$.
- In each module of the r-system, the decisions made by the IAC algorithms form a matrix of $n_w \times k_t$ symbols of which the columns are code words of the code defined by \mathcal{W} . The final decision in each module of the r-system is the outcome of decoding these code words. \square

The algorithm is visualized in figure 3 in the same way as in figure 1.

A subsystem which is connected to a fault-tolerant subsystem according to the pre-observation method can never cause the latter subsystem to go down, regardless of whether the sending subsystem functions correctly or not, i.e.:

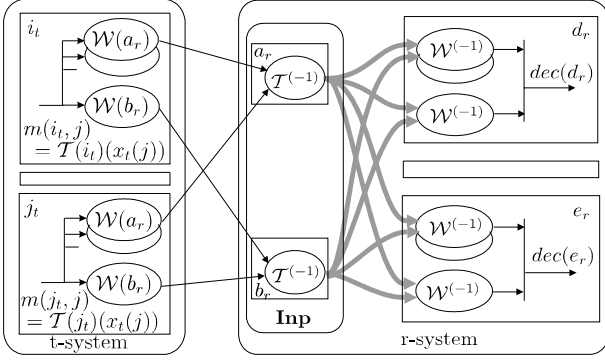


Figure 3: The algorithm based on pre-observation

Theorem 2 *The Input-agreement algorithm based on pre-observation defined in definition 3 satisfies the input-agreement properties defined in definition 1. \square*

Proof:

Recall that the original message x is divided into an array of $k_t \cdot k_w$ symbols $x(i, j)$ of b bits, in which k_t and k_w are the number of data symbols of the T_t -error-correcting code \mathcal{T} and the T_r -error-correcting code \mathcal{W} respectively. From the way in which the original message x subsequently is encoded with \mathcal{T} , \mathcal{W} and thereafter transmitted to the modules of the set **Inp**, we conclude that the $n_t \cdot n_w$ symbols $z(i_t, a_r)$ received by the modules of **Inp** are code words of the product code $\mathcal{T} \times \mathcal{W}$. In terms of the product code, these symbols form an $n_t \times n_w$ matrix. A malfunctioning module i_t might cause a distorted row in this matrix and a malfunctioning receiving module a_r might cause a distorted column. From lemma 2 we know that up to T_t rows and T_r columns can be corrected. From lemma 2 and 1 we conclude that the order in which the decoding functions $\mathcal{T}^{(-1)}$ and $\mathcal{W}^{(-1)}$ are applied, does not matter. From lemma 1 we know that the n_t symbols $z(i_t, a_r)$ received by a module a_r form a possibly mutilated code word of the code defined by \mathcal{T} . So decoding these code words with $\mathcal{T}^{(-1)}$ in each module a_r , will correct all erroneous symbols caused by malfunctioning processors i_t in the t-system, provided the latter contains at most T_t faulty modules and provided a module a_r in functioning correctly.

The k_t symbols obtained in a module a_r after decoding with $\mathcal{T}^{(-1)}$ are denoted by $q'(i, a_r)$, $1 \leq i \leq k_t$. Hence if the t-system is functioning correctly and a_r is correct then, according to lemma 1, $q'(i, a_r) = q(i, a_r)$ in which for each i the symbols $q(i, a_r)$ are a code word of \mathcal{W} which could be obtained by applying the function \mathcal{W} on the symbols $x(i, j)$ of the original value x .

Each module a_r communicates the symbols $q'(i, a_r)$

with $1 \leq i \leq k_t$ to all modules of the r-system by means of an IAC algorithm of the class $\mathcal{A}(T_r, T_r + 1, a_r, \mathbf{Ns}_r)$. This yields in each correct module d_r a number of $k_t \cdot n_w$ decisions $dec((i, a_r)d_r)$ about the symbols $q'(i, a_r)$ communicated to d_r . From the properties of the IAC algorithms we know that if a_r is correct $dec((i, a_r)d_r) = q'(i, a_r)$ and thus if the t-system is also correct $dec((i, a_r)d_r) = q(i, a_r)$. So for each i the decisions $dec((i, a_r)d_r)$ form a code word of \mathcal{W} containing at most T_r erroneous symbols. Decoding these code words with $\mathcal{W}^{(-1)}$ delivers the symbols $x(i, j)$ of the original value x .

So if the t-system is functioning correctly in each correct module d_r , the final decision $dec(d_r)$ equals the original data value x , (property 2 of definition 1).

The agreement property of the IAC algorithm guarantees for any correct d_r and e_r that $dec((i, a_r)d_r) = dec((i, a_r)e_r)$. In all modules the same function $\mathcal{W}^{(-1)}$ is applied and thus $dec(d_r) = dec(e_r)$, (property 1 of definition 1).

This completes the proof of theorem 2. \square

6 Some examples of the interconnection of fault-tolerant systems

In this section we will elaborate on some system parameters imposed by the input-agreement algorithms. We will present a number of simple examples of the interconnection of fault-tolerant systems by means of input-agreement algorithms with $T_t = 0$ or 1 and $T_r = 1$ or 2. For these examples the number of messages, which needs to be transmitted in order to obtain input-agreement on one message, will be presented and the algorithms based on post- and pre-observation will be compared. The examples will elucidate the way in which the parameters are chosen and the way in which the algorithms are constructed.

Post-observation: (example 8 in table 1)

We interconnect two ft-systems both based on masking according to the (4, 2)-concept [5]. So $n_t = n_r = 4$ and $T_t = T_r = 1$. The initial data value x in the t-system is encoded with a single-symbol-error-correcting code \mathcal{T} , with parameters (n_t, k_t, b_t) . From $n_t = 4$, $T_t = 1$, $n_t - k_t \geq 2 \cdot T_t$ and $b_t \geq \log_2(n_t - 1)$, we obtain $k_t = 2$ and $b_t = 2$ (preliminary choice). Each symbol of this code word is encoded with a T_r error-correcting code \mathcal{W}_{i_t} with parameters $(n_w, i_t, k_w, i_t, b_w, i_t)$, in which i_t is the module in the t-system. Choosing different codes for different i_t has no advantage. For efficiency reasons we choose the sets **Inp**(i_t) as large as possible. So **Inp**(i_t) = **Ns**_r and thus a (4, 2, 2) code \mathcal{W} suffices. The data word of this

code consists of two symbols of two bits, so these four bits must correspond to a symbol of the code \mathcal{T} . Hence \mathcal{T} must be a $(4, 2, 4)$ code. For a general method for determining the symbol size we refer to [7, 6].

In total 16 symbols of 2 bits are transferred from the t-system to the r-system for communicating 2 data symbols of 4 bits. So per source messages an amount of 4 messages are transmitted.

The four symbols received by each module a_r in \mathbf{Ns}_r from the four modules of the t-system are communicated to all modules of the r-system by means of algorithms of the class $\mathcal{A}(1, 2, a_r, \mathbf{Ns}_r)$. This class contains only one algorithm, which is defined as follows: The message in a_r is encoded with a $(3, 1, 1)$ code, \mathcal{B} . Each of the symbols of the code word is sent to a different module. In total $3 \times 4 \times 4$ symbols of 2 bit are communicated in this round. During the last round the 48 symbols are forwarded unchanged to all modules (exactly 2) of the r-system which did not yet receive the message. This gives 96 communication. So in total 144 symbols of 2 bits are communicated inside the r-system for communicating 2 data symbols of 4 bits. Per source messages in the t-system, this requires 36 messages to be transmitted inside the r-system.

During the decision making process in each module d_r first the decoder function $\mathcal{B}^{(-1)}$ (majority vote) is applied on the messages received from each module a_r (twelve times in each module). Thereafter the decoder function $\mathcal{W}^{(-1)}$ is applied in order to compensate for faulty modules a_r (four times in each module) and finally the decoder function $\mathcal{T}^{(-1)}$ is applied (only once in each module).

The algorithm is characterized by the codes that are used successively for communicating the initial data in the t-system to all modules of the r-system. For the algorithm in this example this sequence is $[4, 2, 4]_1 [4, 2, 2]_4 t([3, 1, 2]_{16} t f 2_{48} \langle 3, 1, 1 \rangle_{48}) \langle 4, 2, 2 \rangle_{16} \langle 4, 2, 4 \rangle_4$. In which $[n, k, b]$ means encoding with a (n, k, b) code, t means transmission of each symbol of a code word to a different module, $f x$ forwarding unchanged to the remaining x modules and $\langle n, k, b \rangle$ means decoding with a (n, k, b) code. For each encoding and decoding function the number of times the function is applied in the entire system is given in subscript. The sequence of coding and decoding which applies to the IAC algorithm is put between brackets $()$.

Pre-observation: (example 10 in table 1)

In this example the same input-agreement problem as in the previous example will be solved with an algorithm based on pre-observation (section 5). So $n_t = n_r = 4$ and $T_t = T_r = 1$. We choose $\mathbf{Inp} = \mathbf{Ns}_r$ for efficiency. So the product code is built from a

$(4, 2, 2)$ code \mathcal{T} and $(4, 2, 2)$ code \mathcal{W} . The initial source message x thus consists of a 2×2 array of 2 bit-symbols, i.e. 8 bits. After encoding with \mathcal{W} , each module i_t contains 4 symbols of two bit. This yields a total communication to the r-system of 16 symbols of 2 bits, or an amount 4 messages per source message.

In each modules a_r of \mathbf{Inp} the decoder function $\mathcal{T}^{(-1)}$ is applied on the received 4 symbols, which results in two symbols of two bits. The latter two symbols are communicated to all modules of the r-system by means of an algorithm of the class $\mathcal{A}(1, 2, a_r, \mathbf{Ns}_r)$. From the previous example and [7] we know that this takes 9 messages per message in a_r . So in total $4 \times 2 \times 2 \times 9 = 144$ bits are communicated or an amount of 18 messages per source message.

The algorithm is characterized with the sequence: $[4, 2, 2]_2 [4, 2, 2]_4 t \langle 4, 2, 2 \rangle_4 ([3, 1, 2]_{8t}, f 2_{24} \langle 3, 1, 2 \rangle_{24}) \langle 4, 2, 2 \rangle_8$.

The input-agreement algorithm based on pre-observation thus only needs 22 messages to be transmitted for one source message, while the algorithm based on post-observation required 40 messages for one source message.

In table 1 some more examples of input-agreement algorithms based on both post- and pre-observation are presented for varying parameters n_t, T_t, n_r and T_r . In this table m.size denotes the minimum size of the initial value in terms of bits, #mess $t \rightarrow r$ gives the amount of data transmitted from the t-system to the r-system in terms of the size of the initial value. Similarly #mess $r \rightarrow r$ gives the amount of data transmitted inside the r-system per round of data exchange due to the IAC algorithms and #mess total the total amount of data transmission.

7 Discussion

The problem of interconnecting fault-tolerant lockstep synchronous systems is primarily characterized by the way fault-tolerance is implemented in the t-system and in the r-systems, i.e. by the parameters n_t, T_t, n_r, T_r and the observation function \mathcal{T} . Comparing the examples in table 1 we see that for the same parameters and $T_t > 1$, the algorithms based on pre-observation are always superior in terms of communication.

For a single input device there is no difference between pre- and post-observation. Viz., in a non-fault-tolerant t-system, i.e. a single system, the data may be considered as being encoded with an identity function. Hence \mathcal{T} is a $(1, 1, b)$ code. Clearly the decoding function $\mathcal{T}^{(-1)}$ is an identity function too. Identity functions of course may be omitted in the sequence of encoding and decoding function and thus the algorithms based on post- and pre-observation are identi-

Ex-ample	n_t T_t	n_r T_r	me- thod	Coding and decoding sequence	m. size	#mess $t \rightarrow r$ round 1	#mess $r \rightarrow r$ rnd 2+3(+4)	#mess total		
1	1	0	4	1	post	$[1, 1, 1]_1 [3, 1, 1]_1 t([3, 1, 1]_3 t f_{29} \langle 3, 1, 1 \rangle_9) \langle 3, 1, 1 \rangle_4 \langle 1, 1, 1 \rangle_4$	1	3	9+18	30
2	1	0	4	1	post	$[1, 1, 4]_1 [4, 2, 2]_1 t([3, 1, 2]_4 t f_{212} \langle 3, 1, 2 \rangle_{12}) \langle 4, 2, 2 \rangle_4 \langle 1, 1, 4 \rangle_4$	4	2	6+12	20
3	1	0	4	1	pre	$[1, 1, 2]_2 [4, 2, 2]_1 t(1, 1, 2)_4 ([3, 1, 2]_4 t f_{212} \langle 3, 1, 2 \rangle_{12}) \langle 4, 2, 2 \rangle_4$	4	2	6+12	20
4	3	1	4	1	post	$[3, 1, 1]_1 [3, 1, 1]_3 t([3, 1, 2]_9 t f_{227} \langle 3, 1, 2 \rangle_{27}) \langle 3, 1, 1 \rangle_{12} \langle 3, 1, 1 \rangle_4$	1	9	27+54	90
5	3	1	4	1	post	$[3, 1, 4]_1 [4, 2, 2]_3 t([3, 1, 2]_{12} t f_{236} \langle 3, 1, 2 \rangle_{36}) \langle 4, 2, 2 \rangle_{12} \langle 3, 1, 4 \rangle_4$	4	6	18+36	60
6	3	1	4	1	pre	$[3, 1, 1]_1 [3, 1, 1]_3 t(3, 1, 1)_3 ([3, 1, 1]_3 t f_{29} \langle 3, 1, 1 \rangle_9) \langle 3, 1, 1 \rangle_4$	1	9	9+18	36
7	3	1	4	1	pre	$[3, 1, 2]_2 [4, 2, 2]_3 t(3, 1, 2)_4 ([3, 1, 2]_4 t f_{212} \langle 3, 1, 2 \rangle_{12}) \langle 4, 2, 2 \rangle_4$	4	6	6+12	24
8	4	1	4	1	post	$[4, 2, 4]_1 [4, 2, 2]_4 t([3, 1, 2]_{16} t f_{248} \langle 3, 1, 2 \rangle_{48}) \langle 4, 2, 2 \rangle_{16} \langle 4, 2, 4 \rangle_4$	8	4	12+24	40
9	4	1	4	1	pre	$[4, 2, 2]_1 [3, 1, 2]_4 t(4, 2, 2)_3 ([3, 1, 2]_6 t f_{218} \langle 3, 1, 2 \rangle_{18}) \langle 3, 1, 2 \rangle_8$	4	6	9+18	33
10	4	1	4	1	pre	$[4, 2, 2]_2 [4, 2, 2]_4 t(4, 2, 2)_4 ([3, 1, 2]_8 t f_{224} \langle 3, 1, 2 \rangle_{24}) \langle 4, 2, 2 \rangle_8$	8	4	6+12	22
11	1	0	7	2	post	$[1, 1, 18]_1 [7, 3, 6]_1 t([6, 2, 3]_7 [5, 1, 3]_{42} t f_{4210} \langle 5, 1, 3 \rangle_{210} \langle 6, 2, 3 \rangle_{42}) \langle 7, 3, 6 \rangle_7 \langle 1, 1, 18 \rangle_7$	18	$\frac{7}{3}$	7+35+140	184.3
12	3	1	7	2	post	$[3, 1, 18]_1 [7, 3, 6]_3 t([6, 2, 3]_{21} [5, 1, 3]_{126} t f_{4630} \langle 5, 1, 3 \rangle_{630} \langle 6, 2, 3 \rangle_{126}) \langle 7, 3, 6 \rangle_{21} \langle 3, 1, 18 \rangle_7$	18	7	21+105+420	553
13	3	1	7	2	pre	$[3, 1, 6]_3 [7, 3, 6]_3 t(3, 1, 6)_7 ([6, 2, 3]_7 [5, 1, 3]_{42} t f_{4210} \langle 5, 1, 3 \rangle_{210} \langle 6, 2, 3 \rangle_{42}) \langle 7, 3, 6 \rangle_7$	18	7	7+35+140	189
14	4	1	7	2	pre	$[4, 2, 3]_3 [7, 3, 3]_3 t(4, 2, 3)_7 ([6, 2, 3]_7 [5, 1, 3]_{42} t f_{4210} \langle 5, 1, 3 \rangle_{210} \langle 6, 2, 3 \rangle_{42}) \langle 7, 3, 3 \rangle_{14}$	18	$\frac{28}{6}$	7+35+140	186.7

Table 1: Examples of input-agreement protocols based on pre- or post-observation.

cal in case of a single input device.

The classes of input-agreement algorithms based on post- and pre-observation encompass algorithms that have been published earlier, [4, 5]. In particular the examples 1, 6 and 9 are published earlier in [4]. In [4], the idea behind the input-agreement algorithms was as follows: Each module of the t-system sends its data to $2T_r + 1$ modules of the r-system. If the t-system is a fault-tolerant system, the data received by the modules of the r-system is decoded with the observation function $\mathcal{T}^{(-1)}$ of the t-system. Finally the data is distributed to all modules of the r-system by means of an IAC algorithm. Multi-casting data to n modules is the same as encoding with an $(n, 1, b)$ code and thereafter transmitting each symbol of the code word to a different module of the n modules. So the examples in [4] correspond to algorithms based on pre-observation in which for the function \mathcal{W} a $(2T_r + 1, 1, b)$ repetition code is chosen. We have seen that the application of product codes gives more freedom for the choice of \mathcal{W} and thus leads to more efficient algorithms.

References

[1] M.Barborak, M.Malek, "The consensus Problem in Fault-Tolerant Computing" *ACM Computing Surveys*, Vol.25, No 2, June 1993.

[2] D.Dolev and H.R.Strong, "Autenticated algorithms for Byzantine agreement" *SIAM COMP*, Vol.12, No.4, pp.656-666, Nov. 1983.

[3] M.Fischer and N.Lynch, "A lower bound for the time to assure interactive consistency" *Inform. Proc. Letters*, Vol.14, pp.183-186, 1982.

[4] Th.Krol, W.J.van Gils, "The Input/Output architecture of the (4,2) concept fault-tolerant computer" *15-th Annual Symposium on Fault-Tolerant Computing*, pp254-259, Ann Arbor, MI, June, 1985.

[5] Th.Krol, "(N,K) Concept Fault tolerance" *IEEE Tr. on Comp.*, Vol.C35, No 4, April 1986, pp339-349.

[6] Th.Krol, "A Generalization of fault-Tolerance Based on Masking", *Ph.D. dissertation*, Eindhoven University of Technology, Eindhoven, the Netherlands, Sept. 1991.

[7] Th.Krol, "Interactive Consistency Algorithms based on Voting and Error-correcting Codes" *25-th Annual Symposium on Fault-Tolerant Computing*, pp89-98, Pasadena, CA, June, 1995.

[8] F.J.MacWilliams and N.J.A.Sloane, "The Theory of Error-Correcting Codes." Amsterdam, The Netherlands: North Holland, 1978.

[9] M.Pease, R.Shostak, and L.Lamport, "Reaching agreement in the presence of faults" *J. Assoc. Comput. Mach.*, Vol.27, No.2, pp228, 1980.

[10] A.Postma, "Classes of Byzantine Fault-Tolerant Algorithms for Dependable Distributed Systems", *Ph.D. dissertation*, University of Twente, Enschede, the Netherlands, Febr. 1998.