On Synchrony in Dynamic Distributed Systems

Francesc D. Muñoz-Escoí, Rubén de Juan-Marín

Institut Universitari Mixt Tecnològic d'Informàtica
Universitat Politècnica de València
46022 València - SPAIN

{fmunyoz,rjuan}@iti.upv.es

Technical Report TR-IUMTI-SIDI-2018/001

On Synchrony in Dynamic Distributed Systems

Francesc D. Muñoz-Escoí, Rubén de Juan-Marín

Institut Universitari Mixt Tecnològic d'Informàtica
Universitat Politècnica de València
46022 València - SPAIN

Technical Report TR-IUMTI-SIDI-2018/001

e-mail: {fmunyoz,rjuan}@iti.upv.es

June 26, 2018

Abstract

Many modern distributed services are deployed in dynamic systems. Cloud services are an example. They should provide service to a potentially huge amount of users and may require a wide geographical deployment in multiple data centres. Their amount of server processes should change in accordance to the workload variations, showing an adaptive behaviour in order to minimise economical costs.

Dynamic distributed systems may be classified considering two axes: (a) the number of processes that compose the system, and (b) the diameter of the networking graph that interconnects those processes. Other important features of dynamic systems can be derived from these two characteristics, e.g., their attainable synchrony. We analyse the level of synchrony that may be achieved in each dynamic system class and revise the existing techniques for transforming an initially asynchronous large dynamic system into another one with a higher synchrony level. With this, a larger set of problems may be handled in dynamic distributed systems. This facilitates the implementation and provision of additional services in those systems.

KEYWORDS: distributed system, dynamic system, system interconnection, synchrony, failure detector, participant detector.

1 Introduction

Service continuity has traditionally been one of the objectives in many distributed applications. To this end, processes and data should be replicated, but replication, concurrency and failures should be transparent to users [34], since another goal of those distributed applications is to provide a single-system image, i.e., to achieve *distribution transparency*. Unfortunately system components may fail. Failed components may eventually recover and be re-integrated in the system. As a result, a first level of dynamism is introduced in distributed algorithms when a recoverable system model is assumed, since the set of participating processes may evolve along the algorithm execution.

Peer-to-peer collaborative applications [6], where nodes may join and leave the system at will, mobile ad-hoc networks [33] in which the participating elements may temporarily be out of reach, elastic cloud computing services [31] that manage variable and potentially huge workloads, and several kinds of IoT services [23] (e.g., vehicular monitoring and reporting services for driving/routing assistance in smart traffic systems) have shown that multiple types of distributed systems are dynamic. So, a definition for dynamic distributed systems is needed.

Baldoni *et al.* [8] provide that definition with a complementary classification of those systems. Its two key parameters (degree of concurrency and networking graph diameter) condition the characteristics of each possible class of dynamic distributed system. We revise that classification considering in which classes a synchronous or partially synchronous system model may be assumed. This identifies the set of classes where some distributed problems, like consensus, are solvable when the set of participating processes may vary while those dynamic services run. In order to transform an initially asynchronous dynamic system into another one with a higher level of synchrony, two approaches exist: (a) to identify a stable subset of processes, or (b) to organise the system into a hierarchy of interconnectable subsystems. Those approaches make possible the implementation of additional services in dynamic systems, since the applications that provide those services may run in an apparently synchronous environment.

The rest of this paper is structured as follows. Section 2 summarises the existing definition and classification of dynamic distributed systems [8]. Section 3 refines that classification considering synchrony and states some consequences of that study. Section 4 revises some related work. Finally, Section 5 concludes the paper.

2 Classification of Dynamic Distributed Systems

According to Baldoni *et al.* [8], a dynamic distributed system is:

Definition 1 (Dynamic System)**.** *A continually running system in which an arbitrarily large number of processes are part of the system during each interval of time and, at any time, any process can directly interact with only an arbitrary small part of the system.*

This definition provides a characterisation of dynamic distributed systems that is accompanied with a classification of them. Several considerations arise from Definition 1:

C1: Applications developed for dynamic systems are unable to know the identity of all processes that currently belong to their system.

C2: Because of C1, the communication topology in a system cannot be a fully connected graph. This means that algorithms should use a multi-hop communication mechanism in order to reach all processes when messages should be broadcast to all participants.

Since dynamism implies that processes may join and leave the system at will, the concepts of *system run*, *system graph* and *graph sequence* are defined [8].

Definition 2 (System run)**.** *A system run is a total order on the join and leave events issued by processes that respects their real time occurrence order.*

Definition 3 (System graph)**.** *A distributed system can be represented by a graph $G = (P, E)$, where P is the set of processes that compose the system and E is a set of edges (p_i, p_j), representing bidirectional reliable channels connecting processes p_i and p_j.*

System graphs do not need to be fully connected, and the addition or removal of any node or edge generates a different graph. So, these graph updates define a sequence of graphs in a given system run.

Definition 4 (Graph sequence)**.** *Let $\{G_n\}_{run}$ denote the sequence of graphs through which the system passes in a given run. Each $G_n \in \{G_n\}_{run}$ is a connected graph whose diameter can be greater than one.*

Based on these concepts, the classification given by Baldoni *et al.* [8] considers these two dimensions:

- *Number of concurrent entities* (P). Assuming the *infinite arrival model* proposed in [29], these variants can be distinguished:
  - P^b. The number of processes that concurrently belong to the system is bound by a constant b in all system runs.
  - P^n. The number of processes that concurrently belong to the system is bound in each system run, but may be unbound when the union of all system runs is considered.
  - P^∞. The number of processes that concurrently belong to the system in a single run may grow to infinity as time passes.

Number of processes	Network diameter		
	D^b	D^n	D^∞
P^b	$M^{b,b}$	–	–
P^n	$M^{n,b}$	$M^{n,n}$	–
P^∞	$M^{\infty,b}$	$M^{\infty,n}$	$M^{\infty,\infty}$

Figure 1: Dynamic models considering the P and D parameters.

- *Diameter of the interconnecting graph* (D). This parameter models the "*geographical*" dynamism of the system. To this end, $\{D_n\}_{run}$ denotes the set of diameters of the graphs in $\{G_n\}_{run}$. The alternatives to be considered regarding the graph diameter are:
  - D^b, *Bound and known diameter*. The diameter is bound by b and that bounds value is known by the algorithms, i.e.: $\forall D_n \in \{D_n\}_{run}, D_n \leq b$.
  - D^n, *Bound and unknown diameter*. All diameters $\{D_n\}_{run}$ are finite in each run, but the union of all D_n in $\{D_n\}_{run}$ may be unbound. Therefore, an algorithm has no information on the diameter.
  - D^∞, *Unbound diameter*. The diameter may grow indefinitely in a run.

With those parameters, a composed $M^{P,D}$ set of models is defined. Both parameters can assume values b, n and ∞ to indicate their three variants. From the nine possible models, $M^{b,n}$, $M^{b,\infty}$ and $M^{n,\infty}$ are not possible since their diameter is in fact bound by the number of processes, being these bounds value $D_{max} =\mid P \mid -1$. Therefore, as a result, we have six different models that correspond to the following combinations: $M^{b,b}$, $M^{n,b}$, $M^{\infty,b}$, $M^{n,n}$, $M^{\infty,n}$ and $M^{\infty,\infty}$. Those models are depicted in Fig. 1.

3 Classification Refinement

Let us refine in Section 3.1 dimension P from the classification given at [8]. To this end, we present the concurrency subclasses proposed in [1]. Section 3.2 analyses which level of synchrony may be achieved in each class of dynamic distributed system. That analysis may be used for studying in which classes some distributed system problems (e.g., consensus) may be solved. Section 3.3 describes which techniques may increase that inherent level of synchrony in non-synchronous classes. Later on, Section 3.4 discusses what is the place of static distributed systems in that classification.

3.1 Concurrency Refinement

Let S be a distributed system. Marcos K. Aguilera [1] identifies the following process models in distributed systems that may consist of infinitely many processes:

- M_1^n: S has a finite number (n) of processes. This model is called the *n-arrival* model. Algorithms know the n value in this model.
- M_2: S has infinitely many processes, but each run has only finitely many. This model is called the *finite arrival* model. Algorithms do not know how many processes will participate in each run.
- M_3: S has infinitely many processes, runs can have infinitely many processes, but in each finite time interval only finitely many processes take steps. This model is known as the *infinite arrival* model and contains two subsets:
  - M_3^b: An M_3 model where every run has a maximum concurrency bound by constant b (known by algorithms). In this model, there may be infinitely many processes only when processes depart at the same rate that new processes join S. It is known as the *infinite arrival model with b-bound concurrency*. It is the P^b model in [8].

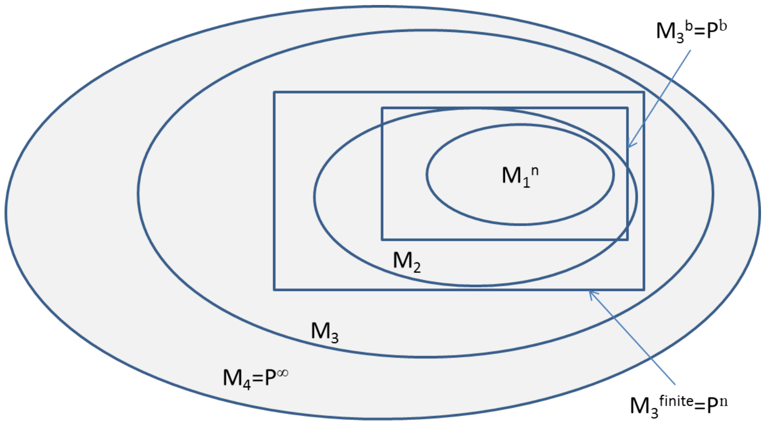

Figure 2: Relations among process system classes.

- M_3^{finite}: An M_3 model in which every run has a maximum concurrency that is finite. It is known as the *infinite arrival model with bound concurrency*. It is the P^n model in [8].

- M_4: S has infinitely many processes, runs can have infinitely many processes, and a finite time interval may have infinitely many processes. This is the *infinite concurrency* model. This is model P^∞ in [8].

Besides, regarding the sets of runs that comply with the conditions of each model, the following model inclusion relationships are identified in [1] (They are depicted in Fig. 2):

- $M_1^n \subset M_2 \subset M_3^{finite} \subset M_3 \subset M_4$
- $M_3^b \subset M_3^{finite}$
- $M_2 \not\subset M_3^b$
- $M_1^n \subset M_3^b$, if $n \leq b$.

These inclusion relations also arise in the P^* models identified in [8], since $P^b = M_3^b$, $P^n = M_3^{finite}$ and $P^\infty = M_4$. Therefore, in order to avoid ambiguity, when we refer in Section 3.2 to a model $M^{\infty,*}$ it should be understood as its set of allowed runs not included in $M^{n,*}$. Similarly, model $M^{n,*}$ refers to its set of runs not included in $M^{b,*}$.

3.2 Achievable Synchrony

Several degrees of asynchrony may be distinguished. To this end, Dolev *et al.* [17] (based on [21]) identify three possible axes of asynchrony in a distributed system:

1. *Processor asynchrony* allows a processor to remain in the same execution step for arbitrary long finite intervals while other processors continue to run.
2. *Communication asynchrony* does not allow to limit message delivery time.
3. *Message order asynchrony* allows messages to be delivered in an order different from the order in which they were sent.

It can be argued that the message order asynchrony is a consequence of the other two axes; i.e., when both processors and communications are asynchronous it is impossible to order the delivery of messages sent by different processors since there is no way to know their concrete sending order.

From the other two remaining axes, communication asynchrony seems to be the principal one. It is accepted that processor synchrony can be simulated with a reasonable effort in a system that uses *logical buffering* [36]; i.e., an algorithm that assumes synchronous processes can be executed using asynchronous processes if the algorithm steps are appropriately numbered in each processor and messages are buffered until their receiver has reached the appropriate step. Communication may be asynchronous to this end. Although there are also general synchronisers (i.e., algorithms that simulate both synchronous processors and synchronous communications) they cannot be easily implemented in a real system (e.g., they require unbound space). As a result, let us revise in the following sections all these dynamic models regarding whether synchronous communication may be achieved in them, since this is a key property in order to decide whether the studied system models could be synchronous or not. Besides, communication is considered *partially synchronous* when there are bounds on message transmission time but the bounds value is unknown [18], and it is considered *synchronous* when the bounds value is known [17].

Let us assume that each point-to-point communication link is synchronous, and δ is its known bound on message transmission time. This assumption provides the best scenario for ensuring that interprocess communication is synchronous, but we will see that even in this ideal scenario most dynamic system models are asynchronous.

According to consideration C2, if any algorithm step requires that a given process (for instance, a coordinator) sends a message to every other process, such message propagation will need *epidemic broadcasting* [24], i.e., each receiver forwards the message to every neighbour and remembers the message identity. Later on, if the same message is received again, it is not resent. Eventually, those messages become propagated to all system processes.

Let us analyse whether processes may assume bounds on message transmission time or not. To this end, there are two dimensions to be considered: process concurrency and network diameter. The degree of process concurrency is more restrictive than the network diameter in what regards communication synchrony. Some P^b models admit an infinite arrival of processes and such infinite arrival rate may delay without any bounds the propagation of messages. On the other hand, the network diameter may only endanger a finite and known message transmission time interval when the $M^{*,n}$ or $M^{\infty,\infty}$ models are considered, but in those models a P^n or P^∞ concurrency model should be used and both admit longer message propagation delays than P^b. Therefore, let us focus our attention on process concurrency models [1]:

- M_1^n (n-arrival) model: There are n processes in the system, and n is known by the algorithms. In this case, the communication paths between those n processes may be found using epidemic broadcasts. Therefore, we may state the following theorem:

  Theorem 1 (M_1^n synchronous communication)**.** *Assuming a δ upper limit on message propagation time through a link, the message transmission time between every pair of processes in a distributed system S that follows the M_1^n model with a bounds value b on the network diameter is bound and its bounds value is known.*

  Proof. Immediate, given the bounds value on the network diameter (b), the known and bound size of the process set (n) and the link transmission time (δ).

  Note that if the network diameter is b, an epidemic broadcast will be able to reach all current system processes in b steps. Since b and δ are known, the resulting bounds value on message transmission time is also known. It is $b\delta$ in the general case, in which there is at least a stable path between p_1 and p_2. □

- M_2 (finite arrival) model: In this model, algorithms do not know how many processes will participate in each run, but it is guaranteed that there is a time t' after which no new processes are started [1]. In this concurrency model, communication is partially synchronous:

  Theorem 2 (Partial synchronous M_2)**.** *The message transmission time between every pair of processes p_1 and p_2 in a distributed system S in model M_2 is bound but that bounds value is unknown.*

  Proof. Let p_1 and p_2 be placed on two opposite edges of the interconnecting network. Let p_{i_1} be the initial process that holds a link between $S - \{p_2\}$ and p_2.

Without loss of generality, let p_k be the single process that directly precedes p_{i_1} in the communication path between p_1 and p_2. (If there were many p_{i_1} processes, what is described hereafter would apply for each one of them, with the same overall result.) Process p_k forwards m to p_{i_1} at time t_0. At time $t_0 + \delta_1$, with $\delta_1 < \delta$, p_{i_1} leaves the system. As a result of this, m is lost. At time $t_0 + \delta_2$, with $0 < \delta_2 \leq \delta_1 < \delta$, p_{i_2} joins the system, defining a path (p_k, p_{i_2}, p_2).

Eventually, p_k detects that m has been lost and it retries at time t_1 to forward m to p_{i_2}. However, at times $t_1 + \delta_1$ and $t_1 + \delta_2$, p_{i_2} is replaced by p_{i_3} and m is lost again. Indeed, those message sending reattempts and process replacements may still occur multiple times from now on. In spite of this, since the process arrival is finite in this model, there will be a time t' after which no other process arrival will happen. At that moment, the links between p_k and p_2 stabilise, and m is finally delivered to p_2. However, we cannot forecast how many reattempts will be done. Thus, the delivery time of m to p_2 is bound, but its bounds value cannot be known. □

- M_3^b (infinite arrival with b-bound concurrency) model: Processes depart from the system at an infinite rate and each time a process leaves the system, another new process replaces that leaving one. Communication is asynchronous in that kind of model, as proven in this theorem:

  Theorem 3 (M_3^b asynchronous communication)**.** *The message transmission time between processes p_1 and p_2 has no bounds in a distributed system S that follows the M_3^b model.*

  Proof. Let us look for a case where the arrival of new processes extends in an unbound way the communication time between p_1 and p_2. Without loss of generality, let us assume that p_1 and p_2 are placed on two opposite edges of the interconnecting network and that a single communication link connects p_2 to the remaining processes in S. Let p_{i_1} be the initial process that holds the unique link between $S - \{p_2\}$ and p_2. (If there were other processes directly linked to p_2, the infinite arrival model admits that what we describe in the following paragraphs for p_{i_1} would be also applicable to all other interconnecting processes when they are receiving m in order to forward it to p_2.)

  Let us imagine that another process p_k in the communication path between p_1 and p_2 is forwarding m to p_{i_1} at time t_0. At time $t_0 + \delta_1$, with $\delta_1 < \delta$, p_{i_1} leaves the system. As a result of this, m is lost. At time $t_0 + \delta_2$, with $0 < \delta_2 \leq \delta_1 < \delta$, p_{i_2} joins the system in a way that it is able to receive the messages sent by p_k and propagate those messages to p_2.

  Eventually, p_k detects that m has been lost and it retries at time t_1 to forward m to p_{i_2}. However, at times $t_1 + \delta_1$ and $t_1 + \delta_2$, p_{i_2} is replaced by p_{i_3} and m is lost again. Indeed, those message sending reattempts and process replacements may still occur infinitely often from now on (in general, $\forall j > 0$, at times $t_j + \delta_1$, $t_j + \delta_2$, being $p_{i_{j+1}}$ replaced by $p_{i_{j+2}}$). Thus, the delivery of m to p_2 is delayed infinitely often. This means that message propagation time has no bounds in this scenario. So, the resulting communication model is asynchronous. □

- M_3^{finite} (infinite arrival with bound concurrency) model: Infinite arrival model in which each run has a maximum concurrency that is finite. Its communication is also asynchronous, as proven in this corollary:

  Corollary 1 (Asynchronous M_3^{finite})**.** *The message transmission time between processes p_1 and p_2 has no bounds in a distributed system S that follows the M_3^{finite} model.*

  Proof. Immediate from Theorem 3. The scenario described in its proof is directly applicable in this model, too, since $M_3^b \subset M_3^{finite}$. □

- M_4 (infinite concurrency) model: Infinite arrival model in which each run has no bounds on its concurrency. Its communication is also asynchronous, as proven in this corollary:

  Corollary 2 (Asynchronous M_4)**.** *The message transmission time between processes p_1 and p_2 has no bounds in a distributed system S that follows the M_4 model.*

Proof. Immediate from Theorem 3. The scenario described in its proof is directly applicable in this model, too, since $M_3^b \subset M_4$. $\square$

Once those concurrency models have been analysed, let us translate those results to the $M^{P,D}$ models proposed in [8]. Section 3.1 has stated those needed translations. With them, the following corollaries state which communication model can be achieved in each $M^{*,*}$ model:

Corollary 3. *The n-arrival $M^{b,b}$ model may assume synchronous communication.*

Proof. Since an n-arrival $M^{b,b}$ system assumes an M_1^n process model, then Theorem 1 directly implies that n-arrival $M^{b,b}$ may assume synchronous communication. $\square$

Corollary 4. *The finite arrival $M^{b,b}$, finite arrival $M^{n,b}$ and finite arrival $M^{n,n}$ models may assume partial synchronous communication.*

Proof. Since all finite arrival systems assume an M_2 process concurrency model, then Theorem 2 determines that all those models may assume partial synchronous communication. Note that Theorem 2 has made no assumptions on the network diameter value. Therefore, its results may be applied in all those models. $\square$

Corollary 5. *The infinite arrival $M^{b,b}$, infinite arrival $M^{n,b}$, $M^{\infty,b}$, infinite arrival $M^{n,n}$, $M^{\infty,n}$ and $M^{\infty,\infty}$ models should assume asynchronous communication.*

Proof. Since all cited systems assume either an M_3^b, M_3^{finite} or M_4 process model, then Theorem 3 (or its corollaries 1 or 2) is applicable in each system. All those cases generate the same result: the asynchronous communication model should be assumed.

Note that Theorem 3 has made no assumptions on the network diameter value. Therefore, its results may be applied in all those models. $\square$

Figure 3 shows graphically which level or levels of communication synchrony correspond to each process concurrency class.

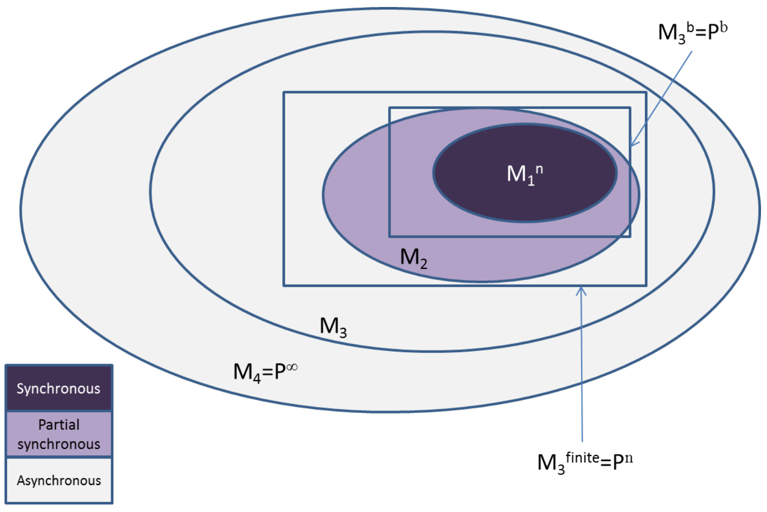

Figure 3: Maximum level of synchrony for each system class.

3.3 Consequences

Our classification refinement implies that some classical problems that are not solvable [20] in traditional asynchronous reliable distributed systems where processes may fail, will not be solvable in $M^{\infty,\infty}$ or in any infinite arrival $M^{*,n}$ or $M^{*,b}$ dynamic systems. This includes *consensus* [21] and many others [20]. On the other hand, problems solvable in (partially) synchronous systems require that the resulting dynamic model becomes one of the finite arrival $M^{n,n}$, $M^{n,b}$ or $M^{b,b}$ subclasses.

Therefore, we need a useful framework for structuring any general (and potentially unbound) dynamic distributed system into a set of smaller and synchronous dynamic subsystems. In this way, traditional algorithms that assume a known number of processes or that need some degree of synchrony could be used in a large dynamic system. In this regard, several basic approaches may be found:

- The first one sets a hierarchical organisation of system processes, defining multiple subsystems and interconnecting them using inter-subsystem channels. Each subsystem uses its own subnetwork and connects with other subsystems using bridges. Therefore, the algorithms in each subsystem see a fully connected network with a logical network graph diameter of length 1. On the other hand, when a process sends a message to another process in a different subsystem, it needs at least one forwarding step driven by some "bridge" process or a path along several bridges. Thus, the global network graph diameter is greater than 1 in that scenario.

  Rodrigues and Veríssimo used those principles in their *causal separators* [32] proposal. Causal separators are a scalability mechanism for causal message multicasting. With them, each subsystem may use its own internal (and different) causal multicast algorithm. Each internally delivered message is forwarded to other subsystems by a specialised bridge process that knows the addresses of the remaining bridges. In order to reach a global causal ordering, such message inter-subsystem forwarding needs FIFO order when there are only two subsystems [32] or causal order in other cases [9]. This provides a first example about how to use a classical algorithm intended for a finite arrival $M^{b,1}$ system (those to be used in each subsystem) combined with another one of the same kind (the interconnecting algorithm, that is also for finite arrival $M^{b,1}$ systems) in order to manage an $M^{n,b}$ system, since the resulting global system has a large and potentially bound but unknown set of processes. This first kind of hierarchical architectures was used in the context of interconnectable message broadcast protocols [32, 9, 26, 15, 16] and interconnectable memory consistency models [19]. A hierarchical organisation provides a solid basis for building large dynamic systems in order to solve problems with a decomposable domain (e.g., support of fast consistency models [7] in case of using replicated data elements, implementation of FIFO or causal message broadcast algorithms [26]...).

  In this first example, the resulting overall system belongs to the $M^{n,b}$ class. Let us revise whether any $M^{*,n}$ system may be handled with this same strategy. To this end, each started process is compelled to be integrated in any of the already existing subsystems. That was the principle that drove partially centralised P2P systems [6]. In those systems, a *supernode* (or *super-peer*) is the only process known by all processes in a given area. Supernodes index all resources in that system subset and they forward the requests that cannot be answered in that subset to other supernodes in order to get an answer. Thus, all system resources may be accessed by every process. The population in each subset varies with time and there are no upper bounds on the global amount of participating processes. In this second example, the resulting system belongs to the $M^{\infty,n}$ class since all supernodes do not know each other: each supernode only needs to know a few other supernodes and their communication is driven by an epidemic broadcast. Besides, each supernode does not need to know the addresses of all the nodes that may use its services. That set of user nodes varies dynamically and cannot be bound nor known with precision. The supernode only knows which resources have been published or downloaded by those processes, i.e., it only knows the addresses of a subset of processes that use its services.

  Another example of global $M^{*,n}$ system that may rely on simpler $M^{*,1}$ algorithms in each subsystem is an interconnectable FIFO multicast algorithm [5]. FIFO multicasting has more relaxed requirements than causal multicasting. Thus, FIFO multicasting only needs FIFO propagation through the inter-bridge channels. If messages are tagged with their initial bridge forwarder identifier in order to avoid repeated delivery, inter-bridge propagation can be achieved using an epidemic propagation. In that case, the bounds on the amount of subsystems that compose the distributed system may be unknown by the participating processes.

- The second approach was proposed by Mostéfaoui *et al.* [30]. It defines a *stable subset* of processes able to ensure algorithm progress. This stable set should comply with some constraints: a minimal number of processes (α) that remain in the system long enough (*stability*) –note that α simulates the static-system requirement of maintaining at least $n - f$ correct processes, where n is the initial number of system processes and f is the current number of failed processes–, and a strong cooperation among those α processes (and this suggests that they assume a fully-interconnecting network among them). Additionally, two complementary communication primitives are provided: a *query-response* that broadcasts a query and waits for α answers, and a *broadcast* operation that is able to propagate information to all system processes. At a glance, this implies that the stable subset conforms to the finite arrival $M^{b,b}$ system model, whilst the overall system may assume even the $M^{\infty,\infty}$ one. Algorithms are executed in an $M^{b,b}$ subpart, propagating their advancements to the remaining processes that may join the distinguished subset if they are sufficiently stable. To this end, they only need to be one of the first α repliers to the *query-response* primitives being executed in the corresponding algorithm.

Fortunately, not all problems demand a synchronous system in order to be solved. So, each problem should be carefully studied to analyse in which kinds of dynamic system it is solvable. A sample of this kind is already presented in [8] where the *one-time query* problem [10] is initially solved only in an $M^{*,b}$ system with the *WildFire* algorithm [10], but its specification is later slightly relaxed in order to build the *DepthSearch* algorithm [8] that solves it also in any $M^{*,n}$ model but not in an $M^{\infty,\infty}$ one.

3.4 Static vs. Dynamic Frontier

Another question that arises from the properties outlined above is where to place the frontier between *static* and *dynamic* systems. According to Definition 1, each process in a dynamic system is unable to know the identity and location of all the remaining processes and that system may have a potentially very large number of processes. Thus, a static system should break both conditions. This means that: (1) the identity and address of all the remaining processes are known by each process, allowing in this way a logically fully connected network among all system processes (i.e., a network graph diameter of length 1), and (2) the amount of processes in the system is bound. This means that we may separate an n-arrival $M^{b,1}$ subclass in the $M^{b,b}$ class shown in Section 2. This new n-arrival $M^{b,1}$ subset corresponds to traditional *static* distributed systems, since the n-arrival concurrency model (i.e., model M_1^n in [1]) guarantees that processes know each other, and a value 1 for the network diameter matches the assumed logical fully connected network.

Therefore, that hypothetical frontier cannot be drawn on a grid of distributed system classes as that depicted in Fig. 1, since static distributed systems are only a subpart of the minimal set that can be defined in the $M^{P,D}$ taxonomy. It is only the n-arrival subset of the $M^{b,1}$ model that can be obtained when the network diameter in class $M^{b,b}$ is set to value 1.

Note that such $M^{b,1}$ model still admits an infinite arrival process concurrency model (i.e., M_3^b in [1]). The dynamic system described in [35] is an example of that model. This shows that $M^{b,1}$ is not entirely static.

4 Related Work

As it has been explained in Consideration C1 and Definition 1, not all processes that belong to a dynamic distributed system know each other. We have analysed the level of synchrony that can be guaranteed in each class of dynamic system. Synchrony may be needed for solving some problems in a distributed system and consensus is an example of those problems. There have been multiple previous papers that have solved consensus with unknown participants. They also studied the needed level of synchrony, although in an implicit way, since the failure detectors [14] being used in each paper were different, and each one requires a given level of synchrony in order to be implemented. For instance, according to Larrea *et al.* [28], $\mathcal{W}$, $\mathcal{Q}$, $\mathcal{S}$ and $\mathcal{P}$ detectors demand a synchronous model, while $\diamond\mathcal{W}$, $\diamond\mathcal{Q}$, $\diamond\mathcal{S}$ and $\diamond\mathcal{P}$ demand at least a partial synchronous model when processes may fail.

To begin with, Jiménez *et al.* proved in [25] that none of the eight failure detector classes originally proposed in [14] may be implemented in a system with unknown membership, but the Ω [13] failure detector can be implemented [2] there. In order to circumvent that impossibility, Cavin *et al.* [11] had previously introduced the concept of *participant detectors*. In that scope, each process calls its local participant detector in order to obtain an approximation on the current set of participating processes. Participant detectors have two properties:

- Information inclusion: The information returned by each detector is non-decreasing over time.
- Information accuracy: A detector never returns a process that does not belong to the system.

In the system assumed in [11] processes cannot fail. That paper proves that consensus may be solved in that system with a *one sink reducibility* (OSR) participant detector. OSR is a participant detector that requires that the detected network graph is connected and that its directed acyclic graph obtained by reducing the original directed interconnection graph to its strongly connected components has only one sink.

In practice, this means that the interconnecting network should be stable and the set of processes respects an M_1^n model [1]. Since failures are not tolerated in that paper, nothing is mentioned about the failure detector being needed for supporting consensus in that scenario or about the actual level of synchrony required for implementing consensus in that system.

In a subsequent technical report [12], the same authors extend their results to a crash-prone model and provide a solution for the consensus problem with unknown participants. In that case, they explicitly recognise that the OSR participant detector should be complemented with a perfect ($\mathcal{P}$) failure detector. According to Larrea *et al.* [28], $\mathcal{P}$ may only be implemented in a synchronous system. This matches what we have outlined in our previous sections, since the system being assumed in [12] is an example of an n-arrival $M^{b,b}$ system and those systems may assume synchronous communication.

Later, Greve and Tixeuil [22] explicitly combine an Ω failure detector with a k-OSR participant detector in order to solve consensus on an asynchronous system with unknown membership. In that case, a finite process set is assumed and processes may fail by crashing. The k-OSR participant detector ensures a k-connected network graph and the system tolerates up to f concurrent failures, with $f < k$. With our results, this corresponds to a finite arrival $M^{b,b}$ model that admits up to a partial synchronous communication model. The usage of an Ω failure detector matches that level of synchrony, according to [28, 25].

Recently, Alchieri *et al.* [4, 3] have addressed the problem of Byzantine consensus with unknown participants for systems with n processes and up to f concurrent failures. The interconnecting network is k-strongly connected, and the algorithm requires a k-OSR participant detector where $f < \frac{k}{2} < n$ and the sink graph component should have at least $3f + 1$ processes. When all those requirements are met, Byzantine consensus may be solved like in any distributed system with known participants. This means that any of the Byzantine-specific failure detectors proposed in [27] may be assumed. Those failure detectors require a partial synchronous system in order to be implemented, according to [27].

5 Conclusions

Baldoni et al. [8] identify six different classes of dynamic systems (specified as $M^{P,D}$), crossing two axes: the degree of concurrency (P) and the network diameter (D). The resulting classes are: $M^{b,b}$, $M^{n,b}$, $M^{\infty,b}$, $M^{n,n}$, $M^{\infty,n}$ and $M^{\infty,\infty}$. We have analysed the highest degree of system synchrony achievable in each class.

Our analysis of the attainable degree of synchrony in every dynamic system class has shown that three classes are inherently asynchronous, even when synchronous links are assumed: $M^{\infty,b}$, $M^{\infty,n}$ and $M^{\infty,\infty}$. Besides, three other subclasses are also asynchronous when they assume an infinite arrival model for their processes: $M^{b,b}$, $M^{n,b}$ and $M^{n,n}$. The algorithms that require at least a partially synchronous model cannot be run in those classes, but there are some system structuring techniques that allow the transformation of asynchronous classes into others that admit a synchronous model. Those techniques are based on two principles: (1) a division of the initial system in multiple interconnectable subsystems with a bound network graph diameter and a global hierarchical structure, imposing a finite arrival model in each subsystem, and (2) the definition of a stable group of processes that sets the needed level of synchrony for allowing algorithm progress. With those techniques, a larger set of services may be implemented in dynamic systems.

The degree of synchrony that may be attained in different classes of dynamic systems has been implicitly studied in previous works focused on solving consensus with unknown participants. In that scope, synchrony is embedded in the implementation requirements of the failure detectors being needed for solving consensus. Our results provide an alternative way of discussing on synchrony in these systems, being more direct than that based on failure detectors.

References

[1] Marcos Kawazoe Aguilera. A pleasant stroll through the land of infinitely many creatures. *SIGACT News*, 35(2):36–59, 2004.

[2] Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. On implementing Omega with weak reliability and synchrony assumptions. In *22nd ACM Symp. Princ. Distrib. Comp. (PODC)*, pages 306–314, Boston, Massachusetts, USA, July 2003.

[3] Eduardo Adílio Pelinson Alchieri, Alysson Bessani, Fabíola Greve, and Joni da Silva Fraga. Knowledge connectivity requirements for solving Byzantine consensus with unknown participants. *IEEE Trans. Dependable Sec. Comput.*, 15(2):246–259, 2018.

[4] Eduardo Adílio Pelinson Alchieri, Alysson Neves Bessani, Joni da Silva Fraga, and Fabíola Greve. Byzantine consensus with unknown participants. In *12th Intnl. Conf. Princ. Distrib. Syst. (OPODIS)*, pages 22–40, Luxor, Egypt, December 2008.

[5] Ángel Álvarez, Sergio Arévalo, Vicent Cholvi, Antonio Fernández, and Ernesto Jiménez. On the interconnection of message passing systems. *Inf. Process. Lett.*, 105(6):249–254, 2008.

[6] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371, 2004.

[7] Hagit Attiya and Roy Friedman. Limitations of fast consistency conditions for distributed shared memories. *Inf. Process. Lett.*, 57(5):243–248, 1996.

[8] Roberto Baldoni, Marin Bertier, Michel Raynal, and Sara Tucci-Piergiovanni. Looking for a definition of dynamic distributed systems. In *9th Intnl. Conf. Paral. Comput. Tech. (PaCT)*, volume 4671 of *Lect. Notes Comput. Sc.*, pages 1–14, Pereslavl-Zalessky, Russia, September 2007. Springer.

[9] Roberto Baldoni, Roy Friedman, and Robbert van Renesse. The hierarchical daisy architecture for causal delivery. In *17th Intnl. Conf. on Distrib. Comput. Syst. (ICDCS)*, pages 570–577, Baltimore, Maryland, USA, May 1997.

[10] Mayank Bawa, Aristides Gionis, Héctor García-Molina, and Rajeev Motwani. The price of validity in dynamic networks. *J. Comput. Syst. Sci.*, 73(3):245–264, 2007.

[11] David Cavin, Yoav Sasson, and André Schiper. Consensus with unknown participants or fundamental self-organization. In *3rd Intnl. Conf. Ad-Hoc, Mobile, and Wireless Networks (ADHOC-NOW)*, pages 135–148, Vancouver, Canada, July 2004.

[12] David Cavin, Yoav Sasson, and André Schiper. Reaching agreement with unknown participants in mobile self-organized networks in spite of process crashes. Technical report, IC/2005/026, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 2005.

[13] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722, 1996.

[14] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.

[15] Rubén de Juan-Marín, Vicent Cholvi, Ernesto Jiménez, and Francesc D. Muñoz-Escoí. Parallel interconnection of broadcast systems with multiple FIFO channels. In *11th Intnl. Symp. on Dist. Obj., Middleware and Appl. (DOA)*, volume 5870 of *LNCS*, pages 449–466, Vilamoura, Portugal, November 2009. Springer.

[16] Rubén de Juan-Marín, Hendrik Decker, José Enrique Armendáriz-Íñigo, José M. Bernabéu-Aubán, and Francesc D. Muñoz-Escoí. Scalability approaches for causal multicast: A survey. *Computing*, 98(9):923–947, September 2016.

[17] Danny Dolev, Cynthia Dwork, and Larry J. Stockmeyer. On the minimal synchronism needed for distributed consensus. *J. ACM*, 34(1):77–97, 1987.

[18] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[19] Antonio Fernández, Ernesto Jiménez, and Vicent Cholvi. On the interconnection of causal memory systems. In *19th ACM Symp. Princ. Distrib. Comp. (PODC)*, pages 163–170, Portland, Oregon, USA, July 2000.

[20] Faith Fich and Eric Ruppert. Hundreds of impossibility results for distributed computing. *Distributed Computing*, 16(2-3):121–163, 2003.

[21] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[22] Fabíola Greve and Sébastien Tixeuil. Knowledge connectivity vs. synchrony requirements for fault-tolerant agreement in unknown networks. In *37th Intnl. Conf. Dependable Syst. and Netw. (DSN)*, pages 82–91, Edinburgh, UK, June 2007.

[23] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (IoT): A vision, architectural elements, and future directions. *Future Generation Comp. Syst.*, 29(7):1645–1660, 2013.

[24] Indranil Gupta, Anne-Marie Kermarrec, and Ayalvadi J. Ganesh. Efficient epidemic-style protocols for reliable and scalable multicast. In *21st Symp. Reliab. Distrib. Syst. (SRDS)*, pages 180–189, Osaka, Japan, October 2002. IEEE-CS Press.

[25] Ernesto Jiménez, Sergio Arévalo, and Antonio Fernández. Implementing unreliable failure detectors with unknown membership. *Inf. Process. Lett.*, 100(2):60–63, 2006.

[26] Scott Johnson, Farnam Jahanian, and Jigney Shah. The inter-group router approach to scalable group composition. In *19th Intnl. Conf. on Distrib. Comput. Syst. (ICDCS)*, pages 4–14, Austin, TX, USA, June 1999.

[27] Kim Potter Kihlstrom, Louise E. Moser, and P. M. Melliar-Smith. Byzantine fault detectors for solving consensus. *Comput. J.*, 46(1):16–35, 2003.

[28] Mikel Larrea, Antonio Fernández, and Sergio Arévalo. On the implementation of unreliable failure detectors in partially synchronous systems. *IEEE Trans. Computers*, 53(7):815–828, 2004.

[29] Michael Merritt and Gadi Taubenfeld. Computing with infinitely many processes. In *14th Intnl. Conf. Distrib. Comput. (DISC)*, volume 1914 of *Lect. Notes Comput. Sc.*, pages 164–178, Toledo, Spain, October 2000. Springer.

[30] Achour Mostéfaoui, Michel Raynal, Corentin Travers, Stacy Patterson, Divyakant Agrawal, and Amr El Abbadi. From static distributed systems to dynamic systems. In *24th Symp. on Reliab. Distrib. Syst. (SRDS)*, pages 109–118, Orlando, FL, USA, October 2005. IEEE-CS Press.

[31] Francesc D. Muñoz-Escoí and José M. Bernabéu-Aubán. A survey on elasticity management in PaaS systems. *Computing*, 99(7):617–656, July 2017.

[32] Luís Rodrigues and Paulo Veríssimo. Causal separators for large-scale multicast communication. In *15th Intnl. Conf. on Distrib. Comput. Syst. (ICDCS)*, pages 83–91, Vancouver, Canada, June 1995.

[33] Patricia Ruiz and Pascal Bouvry. Survey on broadcast algorithms for mobile ad hoc networks. *ACM Comput. Surv.*, 48(1):8:1–8:35, 2015.

[34] Irving L. Traiger, Jim Gray, Cesare A. Galtieri, and Bruce G. Lindsay. Transactions and consistency in distributed database systems. *ACM Trans. Database Syst.*, 7(3):323–342, 1982.

[35] Sara Tucci-Piergiovanni and Roberto Baldoni. Eventual leader election in infinite arrival message-passing system model with bounded concurrency. In *8th European Dependable Computing Conf. (EDCC)*, pages 127–134, Valencia, Spain, April 2010.

[36] Jennifer L. Welch. Simulating synchronous processors. *Inf. Comput.*, 74(2):159–170, 1987.