

How to Overcome the Limits of Bounds*

by

Olivier Peres

Technical Report #2009-08

August 2009

How to Overcome the Limits of Bounds*

Olivier Peres[†]

Department of Computer Science, Ben-Gurion University of the Negev
Beer-Sheva, Israel 84105
olivier@bgu.ac.il

Abstract

A distributed system consists of processes linked to one another by communication channels. A classical problem is how to model these channels realistically so that it is possible to write correct algorithms. Many algorithms are written under the assumption that the channels deliver the messages in the order in which they were sent, and an unknown arbitrary bound is assumed on the capacity of each channel. This stems from the observation of a hardware communication link, but raises the need for the algorithms using the channels to cope with these bounds. This article presents a new solution, called IO-fairness, that results in more natural executions and removes the need for artificial workarounds. IO-fairness can be used on individual algorithms and, also importantly, is useful to execute global algorithms made of several composed subalgorithms.

Keywords: message passing, non-FIFO channels, scheduling.

1 Introduction

Distributed systems consist of *processes* linked to one another by communication channels. They are an abstraction over computer networks, where machines communicate with one another via specialized hardware, like wires and wireless links, themselves linked by routers and other networking equipment, forming a given topology. Various models, and algorithms using them, have been described in the literature [6].

This paper focuses on modeling computer networks, and is therefore about message passing systems. Two major criteria are used to classify message passing systems: the constraints on the *scheduler*, the entity that decides which process(es) can make a move at any given time, and the constraints on the channels.

*A brief announcement of this paper was accepted for publication in SSS 2009.

[†]Partially funded by the ICT program of the European Union under contract number ICT-2008-215270.

Schedulers can be centralized or distributed, the latter being generally preferred for distributed systems since it allows the processes to be activated independently from one another, and they can be more or less *fair*, i.e. more or less severe restrictions can be placed on what they can do. Unfair schedulers are very difficult to cope with, since they typically can prevent some processes from making any progress. On the other hand, a very fair scheduler is unrealistic and makes writing algorithms easier than in real conditions. A common choice is *weak fairness*, defined as follows: if an action is enabled infinitely often, then it is eventually executed.

Channels may deliver the messages in order (FIFO) or not, and their capacity can be bounded or not. It is common to assume bounded FIFO channels, based on the observation that a real-life wire does not reorder messages and has a bounded capacity.

Bounded FIFO channels do have their shortcomings, though. This is illustrated by the method that Afek and Bremler had to design for working around these shortcomings for their *power supply* spanning tree algorithm [1]. They assume that it is possible to detect the message loss that occurs when one tries to send a message into a full channel and set up buffer variables in which lost messages are stored, to be sent again later. However, this hypothesis is stronger than it appears, since it requires observing a link into which no message can be sent.

A New View on Communication Channels

The contribution of this paper consists in promoting a fresh view on communication channels and their relation to scheduling. This new theory is based on the observation that contemporary systems are essentially commuted networks, which means that the “channel” between two processes is, in fact, a set of links, routers and other processes. In this context, the FIFO assumption is unrealistic, because messages in transit from a given process to another process can take any possible route, and thus overtake one another. Assuming a fixed bound on each communication channel is also unrealistic, because if a given hardware link is full, the router on the incoming end will re-route the messages so that they take a different path. However, this does not mean that any behavior becomes realistic. Rather, it means that placing constraints on individual channels is not desirable anymore, and that constraints should be placed on the scheduler instead.

This new approach is called *IO-fairness*, denoting that the scheduler is forbidden to make unrealistic moves (*fairness*) related to the delivery of messages. More precisely, the scheduler cannot store an infinite amount of messages in the channels. In this framework, the channels are unbounded, they are not FIFO, and no attempt is made at making them FIFO, i.e. no data link protocol is used to reorder the messages. IO-fairness ensures that a malicious scheduler cannot take advantage of this to prevent progress.

```

variable: state  $\in \{0,1,2\}$ 
true  $\longrightarrow$  send +; send -; send -
reception of -  $\longrightarrow$  state  $\leftarrow \max(0, \text{state} - 1)$ 
reception of +  $\longrightarrow$  if state  $\neq 0$  then state  $\leftarrow \min(2, \text{state} + 1)$ 

```

Figure 1: +/− Algorithm

Formally, an execution is IO-fair if and only if there exists B such that during the whole infinite execution, no channel contains more than B messages. A scheduler is IO-fair if and only if it only admits IO-fair executions.

This is not equivalent to a bound on the channels in the traditional sense. Firstly, because no individual channel has a fixed bound. This makes it impossible for processes to measure the bounds on their channels, a trick commonly used to turn a non-FIFO channel into a FIFO one. Secondly, because sending a message cannot fail. Since there is no hard bound, the channel always has room to accommodate the message that is sent.

The rest of this paper is organized as follows. Section 2 demonstrates that IO-fairness is realistic using a simple example, the +/− algorithm. Section 3 shows that IO-fairness is enough for a classical routing table algorithm to terminate using unbounded non-FIFO channels and that IO-fairness is a necessary and sufficient condition for *composed* algorithms to work with unbounded non-FIFO channels. Section 4 concludes the paper.

2 Example: the +/− Algorithm

One would expect the +/− algorithm, provided in Figure 1, to work in any realistic system. Each of the two processes in the system has a state in $\{0, 1, 2\}$, initialized arbitrarily. Each process periodically sends to the other the sequence of messages +, −, −. + increments the state of the receiver and − decrements it, however the state cannot be greater than 2 or change if it is 0. In other words, the system should *converge* towards a *legitimate configuration* where all the states are 0, which the system cannot leave. This algorithm is thus *self-stabilizing* [2, 3].

Suppose that the initial state of both processes is 2. If the scheduler is not IO-fair, it can prevent convergence by first almost filling the channel with messages, leaving space for only one or two of them. Then, it lets the +, −, − sequence be sent, so that one or two of the − are lost.

With bounded FIFO channels, one would have to work around this problem by detecting the message losses and compensating for them. On the other hand, an IO-fair scheduler, even though the channels are not FIFO and unbounded, forbids this pathological behavior, since not converging would mean storing an infinite amount of − messages in the channels.

3 Enabling Message-Passing Fair Composition

Algorithms are typically specified, written, proven correct one by one. In practice, however, one often needs to use several algorithms together.

The classical technique that allows this to work is *fair composition*. Let P and Q be distributed algorithms. P performs some task and places the result in its variables, but cannot detect its termination. Q can read the variables of P and *eventually* performs another task under the condition that P terminates, i.e. there is a suffix of the execution in which Q verifies its specification, regardless of what might happen in the beginning of the execution. Then $P \oplus Q$, the global algorithm obtained by merging the code and the variables of P and Q , performs the tasks of P and Q . This technique was introduced for self-stabilizing algorithms [4], but is not limited to them. It is used here with initialized, non-self-stabilizing algorithms.

The first algorithm presented is a classical routing table algorithm. This algorithm, which terminates under the assumption of IO-fairness, can be composed with any algorithm that needs routing tables, provided it is specified in terms of *eventually* solving a task. This yields a global algorithm. The point of this section is to show that, assuming the channels are non-FIFO and unbounded, IO-fairness is necessary and sufficient for the global algorithm to work.

3.1 Routing Table Algorithm

The classical routing table, given in Figure 2, is similar to RIP [5], one of the most widely used algorithms on the Internet. It works as follows. The processes are numbered sequentially from 1 to n . The topology is connected, i.e. there is a path between any two processes. Each process knows n and the numbers of its neighbors. Each process has a vector of distances and a vector of next hops. For process p , a distance d from process k with hop q means that if p needs to send a message to k , it should send it to q , and the message will reach k in d hops.

Theorem 1. *The Routing Table algorithm terminates under the assumption of IO-fairness.*

Proof. First, notice that progress is monotonic: when a process receives a message, the only change that can occur is decreasing a distance and updating the next hop accordingly. Nothing allows a process to accept in its routing table a higher distance to some process than it currently holds.

If the algorithm has not terminated yet, then at least one process p has a suboptimal routing table. This means that the distance that p knows from q is higher than the actual distance in the topology. Thus, there is at least one process p' in a path linking p to q that can improve its routing table by receiving the distances of its predecessor in the path.

<pre> constants: neighbors (list of process numbers) n: number of processes in the system myself: number of the local process variables: distance (vector[1..n] of $\mathbb{N} \cup \{+\infty\}$) hop (vector[1..n] of process number $\cup \perp$) initial values: distance[myself] = 0, others $+\infty$ hop[myself] = myself, others \perp true \longrightarrow send distances to neighbors reception of d from v \longrightarrow for i \leftarrow 1 to n do if d[i]+1 < distances[i] then distances[i],hop[i] \leftarrow d[i]+1,v done </pre>
--

Figure 2: Routing Table Algorithm

Since the guard of the guarded rule that sends distances is *true*, as long as the system has not converged, messages that make the system progress are spontaneously emitted. Since the channels are not FIFO, p' can receive any number of messages that do not make it progress, but they cannot make it regress either. Eventually, however, p' receives one message that makes it improve its routing table. This is because of IO-fairness: since an unbounded amount of messages that make p' progress is produced, the scheduler cannot store them all in a channel, it has to eventually deliver one of them.

This change in the state of p' leaves the system in a new configuration. If there is at least one suboptimal routing table in this configuration, then the same argument applies until termination is reached. \square

3.2 Composition with another Algorithm

Building and maintaining routing tables is useless by itself. The point is to provide the routing tables to a another algorithm that uses them to perform a task, e.g. renaming, topology building or termination detection. As already mentioned, this task has to be specified as an *eventual* property, which may be false initially, but becomes true at some point in the execution.

Consider the global algorithm $P \oplus Q$ obtained by merging all the code and variables of P and Q . If this algorithm is executed in a system where the channels are not FIFO and unbounded, without IO-fairness, even with weak fairness, then it may never terminate because by reordering the messages, the scheduler may deliver the messages of P and not those of Q . By doing so, it verifies weak fairness since in all configurations, the action of delivering a message on each channel is enabled, and it is indeed executed. This shows that message delivery needs some fairness too.

By contrast, with IO-fairness, the scheduler cannot prevent the system from making progress. The reason is that the scheduler cannot store all the messages of any of the composed algorithms in the channels, because they are produced in unbounded amounts, at least as long as termination has not been reached. This is necessary for any algorithm Q that *eventually* performs its task depending on another algorithm P , because Q cannot know whether P has terminated, unless Q is a termination detection algorithm and has itself terminated.

4 Conclusion

This article introduces IO-fairness, a new point of view on communication channels that moves the constraints from the channels to the scheduler, more accurately modeling the reality of a commuted network by preventing the system from exhibiting a pathological behavior, as shown on the $+/-$ algorithm. Precisely, IO-fairness forbids the scheduler to store an infinite amount of messages in the channels.

As illustrated on the routing table algorithm, IO-fairness is sufficient to solve useful tasks using directly unbounded non-FIFO channels instead of bounded FIFO channels, without any workaround or trick to make the channels FIFO. IO-fairness is also shown to be a necessary and sufficient condition for composed algorithms, a very common setting in real systems, to work with unbounded non-FIFO channels.

References

- [1] Y. Afek and A. Bremner. Self-stabilizing unidirectional network algorithms by power-supply. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA97)*, pages 111–120, 1997.
- [2] E. Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the Association of the Computing Machinery*, 17(11):643–644, 1974.
- [3] S. Dolev. *Self-Stabilization*. MIT Press, 2000.
- [4] S. Dolev, A. Israeli, and S. Moran. Self stabilization of dynamic systems. In *Proceedings of the MCC Workshop on Self-Stabilizing Systems, MCC Technical Report No. STP-379-89*, 1989.
- [5] RFC 2453 : RIP (<http://www.faqs.org/rfcs/rfc2453.html>), 1998.
- [6] G. Tel. *Introduction to distributed algorithms*. Cambridge University Press, 1994.