Available online at www.sciencedirect.com
ScienceDirect
Information Processing Letters 107 (2008) 235–239
Information Processing Letters
www.elsevier.com/locate/ipl

Single-bit messages are insufficient for data link over duplicating channels ☆

Kai Engelhardt [a,∗,1], Yoram Moses [b,2]

[a] School of Computer Science and Engineering, The University of New South Wales, and NICTA, Sydney, NSW 2052, Australia
[b] Department of Electrical Engineering, Technion, Haifa, 32000 Israel

Received 12 September 2007; received in revised form 8 February 2008; accepted 1 March 2008

Available online 20 March 2008

Communicated by P.M.B. Vitányi

Keywords: Distributed systems

1. Introduction

Ideal communication channels in asynchronous systems are reliable, deliver messages in FIFO order, and do not deliver spurious or duplicate messages. Single-bit messages suffice to encode and transmit messages of arbitrary finite length over unidirectional channels of this type. When only the FIFO requirement is relaxed (so that messages may be reordered), the same can be achieved over a bidirectional channel. Fekete and Lynch proved that reliable end-to-end communication (data link) is impossible for (fair) lossy FIFO channels without messages containing header information [5]. The results of Wang and Zuck show that, in non-FIFO models with duplication or loss, reliable end-to-end communication is impossible unless the number of different packet types is greater than the number of messages sequences that can be transmitted [8]. We consider the impact of duplication, and prove a result closely related to Fekete and Lynch for a seemingly better-behaved model that we call RELDFI. Namely, we show that no protocol that uses only single-bit messages enables the sender to notify the receiver which of three values it holds, over an asynchronous, bidirectional, reliable, FIFO channel that may duplicate messages. While single-bit protocols can transmit a binary value over a duplicating channel, our result implies that these cannot be composed to implement a data-link layer, without using a larger set of message types. Intuitively, to transmit more complex messages or to implement a data-link layer, messages must encode some additional control information, e.g., in the form of headers or tags. A general theory of composition for this model, in which messages are assumed to have headers, is presented in [3].

This note is devoted to proving the following result. Consider two processes S and R communicating over a

☆ A preliminary version appeared as [K. Engelhardt, Y. Moses, Single-bit messages are insufficient in the presence of duplication, in: A. Pal, A. Kshemkalyani, R. Kumar, A. Gupta (Eds.), 7th International Workshop on Distributed Computing IWDC 2005, in: Lecture Notes in Comput. Sci., vol. 3741, Springer-Verlag, 2005]. Work was partially supported by ARC Discovery Grant RM02036.

∗ Corresponding author.
E-mail addresses: kaie@cse.unsw.edu.au (K. Engelhardt), moses@ee.technion.ac.il (Y. Moses).

1 National ICT Australia is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

2 Work on this paper happened during a sabbatical visit to the School of Computer Science and Engineering, The University of New South Wales, Sydney, NSW 2052, Australia.

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2008.03.010

single bidirectional, finitely-duplicating FIFO channel. Then no protocol P that uses a message set with only two message types (or, equivalently, uses only single-bit messages) in this setting can guarantee to transmit more than two values from S to R. Since data-link layers enable the transmission of all finite sequences of bits, this result yields that no data-link protocol exists in the above model.

Our result is as strong as can be expected, since two values can trivially be transmitted in RELDFI using a one-bit message. Moreover, it is straightforward to show that a message set of size 3 suffices to transmit arbitrary values, as well as infinite sequences of values. (One message can serve as a delimiter.) The Alternating-Bit Protocol transmits arbitrary sequences of bits using 4 different messages in all FIFO models that we consider [1].

2. Preliminary definitions

Processes and local runs. We consider systems consisting of two processes, a *sender*, S, and a *receiver*, R. We let X range over $\{\mathrm{S}, \mathrm{R}\}$ and denote by $\overline{X}$ the other process. Each process X has a set Σ_X of initial states, a message set M_X, a set A_X of *internal actions*, *send actions* $\mathsf{snd}(m)$ for $m \in M_X$, and *deliveries* $\mathsf{dlv}(m)$ of messages $m \in M_{\overline{X}}$ to X. Internal and send actions of X are called *moves*. An *event* (*of* X) is a move of X or a delivery to X.

A *local run of* X is an infinite sequence $x = \langle v, e_0, \ldots\rangle$ where $v \in \Sigma_X$ and the e_i events of X, infinitely many of which are moves of X (and the remaining ones are deliveries to X). This ensures that a local run treats X fairly, and it prevents crash behavior or denial-of-service scenarios. A *run* $r = (s, l, \delta)$ consists of local runs s and l of S and R, respectively, and a *matching function* δ mapping delivery events to send events. More formally, $\delta : \{\mathrm{S}, \mathrm{R}\} \times \mathbb{N} \to \mathbb{N}$ maps pairs (X, j) to indices k where the jth event in X's local run x is a delivery and the kth event in the other local run $\bar{x}$ is a send of the same message. Moreover, δ satisfies:

Interleaving There exists a total ordering of all events in s and l extending the orders of events in s and l such that $\delta(e)$ precedes e, for all e in the domain of δ.

FIFO δ is monotone, i.e., for $j < k \in \mathbb{N}$ if both e_j and e_k are delivery events to X then $\delta(X, j) \leqslant \delta(X, k)$. This prevents re-ordering of messages.

Reliability δ is surjective, in other words, every send is related to at least one delivery. This prevents message loss.

Finite duplication Every send event is related by δ to at most finitely many deliveries. This prevents infinite duplication of messages.

Observe that our assumption that δ is a total function prevents spurious message from being delivered. The model described above, which we call RELDFI, captures reliable FIFO channels that may finitely duplicate messages.

Local states. A *local state* of X is a nonempty finite prefix $x(k) = \langle v, e_0, \ldots, e_{k-1}\rangle$ of a local run $x = \langle v, e_0, \ldots\rangle$ of X. Observe that no information is discarded from the local state of a process over time. Hence, processes have *perfect recall* and thus, in a precise sense, accumulate knowledge as efficiently as possible.[3]

Protocols. A *protocol* P associates with each process a function from that process's local states to its actions. In particular, the behavior of processes is deterministic.[4] A *run of* P is a run $r = (s, l, \delta)$ where, for each process X and $k \in \mathbb{N}$, the $(k + 1)$st event in X's local run $x \in \{s, l\}$ is either a delivery or an occurrence of the action $P(X)(x(k))$ prescribed by the protocol for the preceding local state $x(k)$. These definitions imply, in particular, that processes cannot prevent messages from being delivered to them. They are thus *input-enabled* in the sense of Lynch and Tuttle [7].

Executions. The crux of the proof of our impossibility result will consist of the construction of runs as limits of chains of finite approximations of runs, which we call finite runs. A *finite run of* P is a triple (a, b, β) where a and b are local states of S and R, respectively, and β is a matching function restricted to these local states, that is, it maps delivery events in a and b to send events in b and a, respectively. Moreover, β satisfies the conditions called **Interleaving**, **FIFO**, and **Finite duplication**, but not necessarily **Reliability** stated above, with a, b, and β substituted for s, l, and δ, respectively. One finite run(a', b', β') is a prefix of another (a, b, β) if a' and b' are prefixes of a and b, respectively, and $\beta' \subseteq \beta$. A *chain* is a sequence $(c_i)_{i\in\mathbb{N}}$ of finite runs where c_i is a prefix of c_{i+1} for all $i \in \mathbb{N}$.

---

[3] For the purpose of proving an impossibility result, perfect recall is preferred over a more explicit notion of local state based on variables. Any modifications to a more general form of local state can be simulated based on the protocol, initial state, and messages received [2].

[4] The restriction to deterministic protocols is again motivated by the kind of result we are after. If a non-deterministic protocol P solves a transmission problem reliably then so does any deterministic protocol compatible with P.

A basic property of this model that we shall use later on is:

Lemma 1. *Every finite run can be extended to a run.*

Proof. Let $c = (s, l, \delta)$ be a finite run of P. For $X \in \{\mathrm{S}, \mathrm{R}\}$ let $\langle m_0^X, \ldots, m_{i_X}^X \rangle$ be the sequence of messages sent by $\overline{X}$ in c outside the range of δ (i.e., not yet delivered in c). Define[5] $c' = (s \cdot \tau_\mathrm{S}, l \cdot \tau_\mathrm{R}, \delta \cup \delta_\mathrm{S} \cup \delta_\mathrm{R})$, where τ_X is $\langle \mathsf{dlv}(m_0^X), \ldots, \mathsf{dlv}(m_{i_X}^X) \rangle$ and δ_X matches the kth of these deliveries to the kth unmatched send of $\overline{X}$ in c. Construct the run r as the limit of the sequence of finite runs $(c_i)_{i \in \mathbb{N}}$ defined as follows. Let $c_0 = c'$ and obtain c_{k+1} inductively from c_k by having each process make the move prescribed by P, and if that move is a send event then a delivery of this message appears immediately after the current move of the other process. The limit r of the c_i is indeed a run of P. □

Knowledge. For a given protocol P, we can talk about what processes *know*[6] with respect to P by considering the set of all runs of P. Specifically, we say that the receiver *knows the sender's initial value*, denoted by $\mathsf{K}_\mathrm{R}\mathsf{v}$, at a local state b (with respect to P) if there exists a value $v \in \Sigma_\mathrm{S}$ such that in every run of P in which the state b appears, the sender's initial state is v. Thus, the fact that R is in state b implies that the sender's value is necessarily v. We say that a protocol P *transmits n values* if $|\Sigma_\mathrm{S}| = n$ and in every run of P the receiver eventually knows the sender's initial value. Formally, this is expressed as: for all runs $r = (s, l, \delta)$ of P in which S starts with an arbitrary element of Σ_S as an initial value, and R starts with initial value λ, there exists $k \in \mathbb{N}$ such that for all runs $r' = (s', l', \delta')$ of P satisfying $l(k) = l'(k)$ we have that $s(0) = s'(0)$.

An intuitive property we now prove is that if there are at least two initial values for the sender, then $\mathsf{K}_\mathrm{R}\mathsf{v}$ requires successful communication:

Lemma 2. *Assume that $|\Sigma_\mathrm{S}| \geqslant 2$, and let $r = (s, l, \delta)$ be a run of P. If $\mathsf{K}_\mathrm{R}\mathsf{v}$ holds at $l(k)$ then $l(k)$ contains a delivery.*

Proof. Let $r = (s, l, \delta)$ be a run and let $k \in \mathbb{N}$ such that $l(k)$ does not contain a delivery. Notice that $l(k)$ is uniquely determined by k. For each $v \in \Sigma_\mathrm{S}$, construct the finite run, $c^{(v)} = (s^{(v)}, l^{(v)}, \delta^{(v)})$ by performing k moves for the sender and the receiver but without delivering a single message should any be sent. Each $c^{(v)}$ can be extended to a run by Lemma 1. Observe that each receiver state $l^{(v)}$ equals $l(k)$. It follows that $\mathsf{K}_\mathrm{R}\mathsf{v}$ does not hold at $l(k)$. □

3. Main result

We are now ready to prove our main result:

Theorem 3. *If $|M_\mathrm{S}| = 2$ then no protocol can transmit 3 values in* RELDFI.

Proof. Let $|\Sigma_\mathrm{S}| = 3$ and $M_\mathrm{S} = \{0, 1\}$. Fix a protocol P and assume, by way of contradiction, that P transmits three values. All finite runs and runs mentioned will be ones of P. A delivery event e to R in a run $r = (s, l, \delta)$ of P is called an *alternation* either if it is the first delivery to R or if its content is distinct from that of the preceding delivery to R. We also call a send event by S an *alternation* if the earliest delivery matched to it is an alternation. In particular, the first send by S and the first delivery to R are alternations. We construct a pair of chains $(c_i)_{i \in \mathbb{N}}$ and $(d_i)_{i \in \mathbb{N}}$ of finite runs of P with different initial sender states but identical local states for R in each pair (c_i, d_i). Let $i \in \mathbb{N}$ and let l_i be R's local state in both c_i and d_i. Since c_i and d_i are finite runs, each of them can be extended to a run by Lemma 1. Since the sender has different initial states in these runs, $\mathsf{K}_\mathrm{R}\mathsf{v}$ does not hold at l_i. As we shall show, the limit of at least one of these chains is a run. In that run the sender's value is never transmitted, contradicting the assumption that P transmits three values.

Outline of the proof. Our first step is to find two values for which the first message sent by the sender is the same. Then, we generate the two chains $(c_i)_{i \in \mathbb{N}}$ and $(d_i)_{i \in \mathbb{N}}$ of finite runs starting from these two sender values, respectively. The intuition underlying the second step is as follows. We maintain an invariant that in c_i and d_i the receiver has the same local state and is scheduled to move at the same local states (which will occur at odd steps of our construction). Since the protocol P is deterministic, R performs the same actions in both chains. Moreover, every message sent by R is delivered immediately. More delicate is the handling of the sender S, whose moves occur at even steps of the construction. If P prescribes the same move for S in both finite runs, then this move is taken, and, if the move is a send, the message is delivered to R. If S is prescribed a send, in one finite run, of a message m that

[5] Given two sequences σ and τ, we use $\sigma \cdot \tau$ to denote the result of appending τ at the end of σ.

[6] Our notion of knowledge here coincides with the formal notion of knowledge in the sense of [6,4].

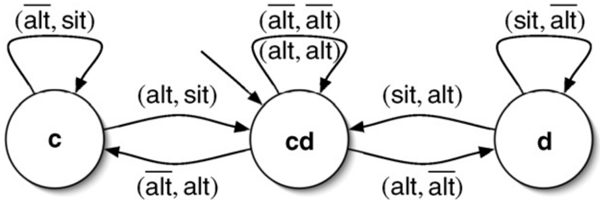

Fig. 1. The construction automaton A.

repeats the most recent message delivered to R, then this message is delivered to R and is regarded by δ as a duplicate delivery in the finite run in which the message was not sent. Finally, if S should send an alternation in one of the finite runs (say c_i) but not in the other, then this message is delayed and the sender is suspended in the corresponding (say c) chain. From this point on, in even steps of the construction, S moves only in the finite runs in which it is not suspended (d), until an alternation is sent by S there. In case this never happens, the limit of the chain in which S continues to move is a legal run in which the value is never transmitted. Indeed, S is guaranteed to move infinitely often in at least one of the chains (possibly both), and such a chain will yield the desired contradiction. To make the above intuition precise, we shall use a simple automaton to help determine in which of the chains S should move at even steps of the construction.

Step 1: Fix $\lambda \in \Sigma_{\mathrm{R}}$. This will be R's initial state in all finite runs and runs considered from now on. For each of S's three initial states, we start a finite run of P and stop it as soon as S sends its first message. Until then, both S and R move in lock step. Every message sent by R in, say, step k is delivered to S right after its kth move. We claim that the sender eventually sends a message in each of these finite runs. Assume by way of contradiction that in one such finite run e the sender does not send any messages. Observe that e contains infinitely many moves by both processes and every message sent is delivered. Thus e is a run. By Lemma 2, however, $\mathsf{K}_{\mathrm{R}}\mathsf{v}$ never holds in e and hence the value is not transmitted. Since the messages sent by S are single bits, in the finite runs starting from at least two of the three values, say v and w, the first message sent by S is the same.

Step 2: Next we construct two chains of finite runs c_i and d_i with initial sender values v and w, respectively. In each step i of the construction, we define two finite runs, $c_i = (s_i, l_i, \delta_i)$ and $d_i = (s'_i, l_i, \delta'_i)$ in which R's initial state is λ. Initially, $s_0 = \langle v\rangle$, $s'_0 = \langle w\rangle$, $l_0 = \varepsilon$, and $\delta_0 = \delta'_0 = \emptyset$. The whole construction is symmetric. We focus on constructing c_i. We distinguish the construction of odd-numbered steps from that of even-numbered ones:

Odd-numbered steps. A step $i = 2k + 1$ of the construction contains a move by R. If that move is a send then the step also contains a delivery of that message to S. More formally, let $e = P(\mathrm{R})(l_{i-1})$. Define $l_i = l_{i-1} \cdot \langle e\rangle$. If e is not a send then $s_i = s_{i-1}$ and $\delta_i = \delta_{i-1}$. Otherwise, if e is $\mathsf{snd}(b)$ then $s_i = s_{i-1} \cdot \langle \mathsf{dlv}(b)\rangle$ and $\delta_i = \delta_{i-1} \cup \{(\mathrm{S}, |s_i|) \mapsto |l_i|\}$.

Even-numbered steps. A step $i = 2k + 2$ of the construction handles a move by S. In this case, however, S might perform a move in just one of the finite runs, or in both. Who moves and how is determined by an auxiliary 3-state automaton and by P. The state σ_i of the automaton in step i is one of **c**, **d**, and **cd**, where the occurrence of a letter in a state's name indicates that the sender moves in the corresponding finite run. (See Fig. 1.) For instance, if $\sigma_{i-1} = \mathbf{c}$ then the sender only moves between c_{i-1} and c_i but not between d_{i-1} and d_i. The initial state of the automaton is **cd**. Odd moves do not affect the automaton state, i.e., $\sigma_{2k+1} = \sigma_{2k}$ for all k.

It is convenient to consider the sender's behavior z in the step from c_{i-1} to c_i, depending on $e = P(\mathrm{S})(s_{i-1})$ and σ_{i-1}, to be one of $\{\mathsf{alt}, \mathsf{skip}, \mathsf{rpt}, \mathsf{sit}\}$. Intuitively, alt stands for the receipt of an alternation; skip stands for an internal action not involving communication; rpt indicates the receipt of a message that is not an alternation; sit means that this sender does not participate in the current step. If $\sigma_{i-1} = \mathbf{d}$ then $z = \mathsf{sit}$. Otherwise we define z as follows. If $e = \mathsf{skip}$ then $z = \mathsf{skip}$. If $e = \mathsf{snd}(b)$ and this send is an alternation then $z = \mathsf{alt}$. Otherwise this send repeats the preceding message, whence we define $z = \mathsf{rpt}$. We define z' based on $e' = P(\mathrm{S})(s'_{i-1})$ and σ_{i-1} analogously.

The transition function of the automaton is described in Fig. 1. Its transitions are labeled with pairs, the first component of which describes z and the second describes z', where $\overline{\mathsf{alt}}$ stands for skip or rpt.

We can now specify the ith step of the construction based on z, z' and σ_{i-1} as follows.

- If $z = \mathsf{sit}$ then $s_i = s_{i-1}$ and otherwise $s_i = s_{i-1} \cdot \langle e\rangle$.

- If $\sigma_{i-1} = \mathbf{cd}$, $z = z' = \mathsf{alt}$, and $e = \mathsf{snd}(b)$, then the alternation is delivered immediately in both chains, that is, $l_i = l_{i-1} \cdot \langle \mathsf{dlv}(b) \rangle$, $\delta_i = \delta_{i-1} \cup \{(\mathrm{R}, |l_i|) \mapsto |s_i|\}$, and δ'_i is obtained analogously.
- If $\sigma_{i-1} = \mathbf{cd}$ and $z = \mathsf{alt}$ but $z' \neq \mathsf{alt}$ then the alternation is not delivered immediately but the sender is suspended from making moves in the following s_j by the automaton entering state $\mathbf{d}$. As long as no alternation is encountered in the following s'_j, the automaton state $\mathbf{d}$ is preserved. When a matching alternation occurs, the pending message is finally delivered, as is the matching alternation, and the automaton returns to $\mathbf{cd}$. Formally, if $\sigma_{i-1} = \mathbf{cd}$, $z = \mathsf{alt}$, and $z' = \mathsf{skip}$, then $l_i = l_{i-1}$ and $\delta_i = \delta_{i-1}$.
- If $\sigma_{i-1} = \mathbf{d}$, $z' = \mathsf{alt}$, and $e' = \mathsf{snd}(b)$ then $l_i = l_{i-1} \cdot \langle \mathsf{dlv}(b) \rangle$ and δ_i will reflect the delivery of the pending alternation, i.e., $\delta_i = \delta_{i-1} \cup \{(\mathrm{R}, |l_i|) \mapsto |s_j|\}$, where $j < i$ is the last step in which S moves between s_{j-1} and s_j. Moreover, $\delta'_i = \delta'_{i-1} \cup \{(\mathrm{R}, |l'_i|) \mapsto |s'_i|\}$.
- If $z = \mathsf{skip}$ and $z' \in \{\mathsf{sit}, \mathsf{skip}\}$ then both $l_i = l_{i-1}$ and $\delta_i = \delta_{i-1}$.
- Suppose that $z = \mathsf{rpt}$ and $e = \mathsf{snd}(b)$. This rpt will be delivered to R immediately, and so $l_i = l_{i-1} \cdot \langle \mathsf{dlv}(b) \rangle$ and $\delta_i = \delta_{i-1} \cup \{(\mathrm{R}, |l_i|) \mapsto |s_i|\}$.
- If $z' = \mathsf{rpt}$ but $z \neq \mathsf{rpt}$ then δ_i will reflect the delivery of a duplicate, that is, $\delta_i = \delta_{i-1} \cup \{(\mathrm{R}, |l_i|) \mapsto |s_j|\}$, where $j < i$ is the last step in which S performs a $\mathsf{snd}()$ action between s_{j-1} and s_j.

The description is complete when taking symmetry into account, swapping the roles of s_i, δ_i, e, $\mathbf{d}$, and z with s'_i, δ'_i, e', $\mathbf{c}$, and z', respectively.

Let $c = \lim_{i\to\infty} c_i$ and $d = \lim_{i\to\infty} d_i$. Observe that the construction has established the following properties.

(1) Both c_i and d_i are finite runs of P, for every i.
(2) The receiver moves infinitely often in both c and d.
(3) The sender moves infinitely often in at least one of c and d, because every even-numbered step of the construction contributes such a move to at least one of them.
(4) All messages sent by the receiver are delivered.
(5) In both, c and d, if the sender moves in step i then all messages it sent earlier have been delivered. It follows that once the sender performs infinitely many moves, all of its messages are delivered.

It follows that at least one of c and d is a run of P and in that run the sender's value is never transmitted. □

An immediate conclusion from Theorem 3 is:

Corollary 4. *No data-link protocol with $|M_S| = 2$ exists in* RELDFI.

Acknowledgements

We thank Alan Fekete and the referees for useful comments on earlier drafts.

References

[1] K.A. Bartlett, R.A. Scantlebury, P.T. Wilkinson, A note on reliable full-duplex transmission over half-duplex links, Communications of the ACM 12 (5) (1969) 260–261.
[2] B. Coan, A communication-efficient canonical form for fault-tolerant distributed protocols, in: Proc. 5th ACM Symp. on Principles of Distributed Computing, ACM Press, 1986.
[3] K. Engelhardt, Y. Moses, Safe composition of distributed programs communicating over order-preserving imperfect channels, in: A. Pal, A. Kshemkalyani, R. Kumar, A. Gupta (Eds.), 7th International Workshop on Distributed Computing IWDC 2005, in: Lecture Notes in Comput. Sci., vol. 3741, Springer-Verlag, 2005.
[4] R. Fagin, J.Y. Halpern, Y. Moses, M.Y. Vardi, Reasoning About Knowledge, MIT Press, 2003.
[5] A. Fekete, N.A. Lynch, The need for headers: An impossibility result for communication over unreliable channels, in: J.C.M. Baeten, J.W. Klop (Eds.), CONCUR '90: Theories of Concurrency: Unification and Extension, in: Lecture Notes in Comput. Sci., vol. 458, Springer-Verlag, 1990.
[6] J.Y. Halpern, Y. Moses, Knowledge and common knowledge in a distributed environment, Journal of the ACM 37 (3) (1990) 549–587.
[7] N.A. Lynch, M. Tuttle, An introduction to input/output automata, CWI Quarterly 2 (3) (1989) 219–246.
[8] D.-W. Wang, L.D. Zuck, Tight bounds for the sequence transmission problem, in: PODC'89: Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, ACM Press, 1989.