A Formal Framework to Reduce Communications in Communication Systems*

Manuel Núñez, Ismael Rodríguez, and Fernando Rubio

Facultad Informática, Universidad Complutense de Madrid
C/. Juan del Rosal, 8, E-28040 Madrid, Spain
{mn,isrodrig,fernando}@sip.ucm.es

Abstract. In this paper we present a method that, given a communicating system, allows us to reduce on average the number of messages sent. The basic idea is to *shortcut* the most frequent *dialogs* appearing in the communications. By doing so, some large dialogs can be reduced to only two messages. In addition, not only the number of messages, but also the size of them, can be minimized, on average, by considering the probabilities of sending each message in each state.

1 Introduction

Compression algorithms are useful to reduce the size of huge amounts of data. In this paper we are only interested in those compression algorithms that are *reversible*, that is, it is possible to recover the original data from the compressed version. This is the case of algorithms as Huffman codes, LZ, and LZW (see e.g. [4,7,6]) which are used in most data compression tools. Even though compression/decompression is usually a computationally costly process, it is often worth to sacrifice this computing capacity because of the limited storing capacity of computing systems. Let us remark that compression algorithms are not only useful to *enlarge* storing capacity. In fact, and this is the use of compression that we consider in this paper, compressed data is getting nowadays very popular to overcome transmission limitations through the Internet.

In this paper we consider communications between distant partners where computational power is *cheaper* than transmission power. Examples of these systems are partners communicating through the Internet or located in a network. A first approach consists in compressing isolated messages, that is, compression at the *message level* (to the best of our knowledge, this is the only method studied in the literature). In this situation, each separate message is independently compressed. We propose a more elaborated reversible method where compression is applied in such a way that not only isolated messages are taken into account but *whole dialogs*. Obviously, in order to compress a dialog we should know it before the transmission starts. This requires that each partner knows in advance what the corresponding communicating partner is going to answer to a given message, and again for the answer to the answer of the first message,

* Work supported in part by the projects TIC2003-07848-C02-01 and PAC-03-001

and so on. Unfortunately, all this information can be known with certainty only when the whole dialog has finished. But it is completely useless to worry about compressing information that has been already transmitted!

Even though we cannot know beforehand a future dialog, we may try to *guess*, with a certain probability, that dialog. For example, let us suppose that P_1 is going to send the message a to P_2. In addition, P_1 could decide that if P_2 replies b then the message c will be sent to P_2. Moreover, it could have already decided (even before the first message is sent) that if P_2 replies with d then e will be sent. Thus, P_1 knows that the complete dialog will be *abcde assuming* that P_2 will reply b to a and d to c. Let us also suppose that we can deduce that the dialog *abcde*, between P_1 and P_2, has a high probability[1]. In this case, it is obvious that it will be profitable to code the whole dialog in such a way that it is not necessary to deliver five messages. In this case, the number of transmitted messages is reduced in a high number of situations (because of the high probability of the corresponding dialog).

Thus, assuming the previous conditions, how does one *anticipate* dialogs? Let us consider again the previous simple example. Let us suppose that P_1 observes that the previous conditions hold, that is, that P_1 decides to send a to P_2, that if afterwards b is received then c will be sent, etcetera. So, P_1 will ask P_2 whether it is willing to perform the whole dialog σ, where both partners know in advance that $\sigma = abcde$. If the dialog σ is compatible with its situation then P_2 will reply *yes*. Otherwise, it will reply with a different message (e.g. b') and the dialog will be continued from that moment on. Let us remark that in the first case we have sensibly reduced the number of messages while in the second case we have not incurred in any lost. Moreover, our approach does not only reduce the size of dialogs by reducing the number of transmitted messages. In fact, it also reduces the size in bits of the most common messages. Thus, the average size of dialogs is reduced, although some low-probability dialogs can be increased.

Following these intuitions, we propose a formal method that, given a communicating system, allows us to compress the most frequent dialogs. In order to know which are the most frequent dialogs, log files recording the past communications can be used. However, they can also be computed by using specifications including the probabilities of sending each message in each state of the machines. We will use this probabilistic information to find out which dialogs should be compressed to decrease the average length of communications in a system. Besides, we will provide a method to compress them by introducing *shortcuts*. Let us remark that our notion of minimizing protocols communication is completely different to the idea of minimizing the number of states of a given protocol, issue that has been extensively studied in the literature (see e.g. [3]).

The rest of the paper is structured as follows. In Section 2 we introduce the basic definitions related to our notion of *dialog finite state machines*. Then, in Section 3 we define how to reduce their communications by introducing *shortcuts*. Finally, in Section 4 we present our conclusions and some lines of future work.

[1] This high probability can be deduced either from the historic record of dialogs or from the specification of the possible communications between the partners

2 Dialog Finite State Machines

In order to apply our method, we need to use a formalism to specify the corresponding systems. Finite state machines have already been considered in the literature for specifying protocols and communicating systems (see e.g. [2,1,5]). Following this idea, in this paper we suppose that the partners of the studied system are formally defined as finite state machines[2]. Thus, by composing them we can get a whole view of the system. This composite view of the system will be formally defined by means of *dialog finite state machines*.

Intuitively, a dialog finite state machine, in short `DFSM`, represents all the possible communications among the components. In every `DFSM`, states can be classified into two classes, depending on the partner that has to send a message. In this sense, we assume a strict alternation between the partners while sending messages. Thus, in any `DFSM` there are only transitions from states associated with a partner to states associated with another partner. Let us remark that a transition represents that a given message has been sent. For example, if the origin of the transition is a state associated to partner P_1 then the destination must be a state associated to P_2. In addition, each transition of the `DFSM` has attached a probability p. Its meaning is that the probability of sending the message attached to that transition is p.

In our notion of `DFSM` we will consider some special messages representing *compound* messages. A message $(m_1 \cdots m_n?)$ represents that one of the partners is proposing to perform a predefined dialog containing n *atomic* messages. A message $(m_1 \cdots m_n!)$ represents that the other partner accepts to perform such a dialog. When this is the case, the whole n-steps dialog is communicated by using only two messages (the proposal and the acceptance). In case the dialog is not to be completely accepted, a message $(m_1 \cdots m_i!b)$ is sent, where $m_1 \cdots m_i$ is the accepted prefix of the dialog, and b is the first message sent after rejecting the rest of the dialog. Finally, if the first partner desires to send m_1 but not to perform the whole dialog $m_1 \cdots m_n$, it can send the message $(m_1 \cdots m_n\#)$. This message allows a partner to distinguish between communicating $m_1 \cdots m_n$ and communicating any other message beginning by m_1 when the compound message $(m_1 \cdots m_n?)$ is available. Let us remark that if $(m_1 \cdots m_n\#)$ is communicated then we do not send only m_1. In fact, we also know that the partner cannot perform the whole dialog. Thus, we can recompute the probabilities of each transition. In order to ease the presentation, we consider systems composed of two partners. From the technical point of view, the extension of our formalism to deal with n-partner systems is trivial (but cumbersome).

In the next definition we introduce the notion of dialog finite state machine. In this definition, and during the rest of the paper, we use the trick of indexing some related sets by i and $3-i$. Let us note that if $i = 1$ then $3-i = 2$, and $i = 2$ implies $3 - i = 1$. Since our communicating partners will be denoted by indexes 1 and 2, this trick will be used to relate each partner with its counterpart.

[2] Actually, we only require that formalizations concern the portion of the behavior of each partner where we want to apply our methodology

Definition 1. A *Dialog Finite State Machine*, in short DFSM, is a tuple $D = (S_1, S_2, M_1, M_2, T_1, T_2)$ where S_i, M_i, and T_i are the sets of *states*, *messages*, and *transitions* of the partner i.

Each message set M_i is defined as a disjoint union $M_i = \alpha_i \cup \beta_i$, where α_i is the set of *atomic messages* and β_i is a set of *compound messages*. Each compound message $m \in \beta_i$ is a sequence of atomic messages of either the form $(m_1 \cdots m_{n-1}!m_n)$ or $(m_1 \cdots m_n\$)$, where for any $1 \leq j \leq n$, if $odd(j)$ then $m_j \in \alpha_i$, while if $even(j)$ then $m_j \in \alpha_{3-i}$. The symbol $\$ \in \{?, !, \#\}$ indicates that the sequence is a *proposal*, a (partial or full) *acceptation*, or a *not-proposal*.

Each transition $t \in T_i$ is a tuple (s, m, p, s') where $s \in S_i$, $s' \in S_{3-i}$ are the initial and final states, $m \in M_i$ is the message communicated in the transition, and $0 < p \leq 1$ is the probability of taking the transition t at state s. Each transition $t = (s, m, p, s')$ will also be denoted by $s \xrightarrow{m}_p s'$.

Let $\texttt{Tran}(s)$ denote the set of transitions (s, m, p, s') departing from state s. Then, for any state $s \in S_i$ we have $\sum\{p \mid (s, m, p, s') \in \texttt{Tran}(s)\} \in \{0, 1\}$, where this value is 0 only when the set is empty. Besides, for any state $s \in S_i$ and message $m \in M_i$ there exists at most one transition $t \in T_i$ such that $t = (s, m, p, s')$. □

Let us comment on the previous definition. First, let us remark that any state belonging to S_i represents that the next partner communicating in the system is partner i. In that state, the communications that i can perform are given by the transitions in T_i departing from that state. Transitions are labelled by both the message transmitted in the communication and the probability of performing such a transition. Let us note that probabilities of all transitions departing from the same state must add 1 (unless there is no transition leaving that state). Simple and compound messages are dealt equally, and they are both transmitted by single transitions. Let us note that any transition performed by partner i leads a DFSM to a state in S_{3-i}, that is, a state where the next communication will be performed by partner $3 - i$. It is worth to point out that this restriction does not decrease the expressivity of the system. For instance, if we want a partner to be able to transmit the sequence of messages ab, then we can consider a single message c representing ab.

Example 1. Let us consider a communication protocol that describes the communications that take place in a peer to peer environment. Specifically, the protocol denotes the negotiation process that takes place when a peer asks another peer to share and transfer some file. The DFSM associated to this protocol is depicted in Figure 1. We suppose that the sharing negotiation process is performed by autonomous agents, where each of them behaves on behalf of its respective user and its preferences. Sharing is a voluntary act, but agents remember previous negotiations, so their behavior with other agents can change depending on their previous decisions (e.g., if an agent A refuses to share files with other agent B, then agent B will punish agent A by refusing to share files with A in the future). Moreover, after a refusal an agent can complain to communicate the other agent that if the request is not accepted then the future sharing with this agent will suffer. Our peer to peer environment includes also some non-standard features.

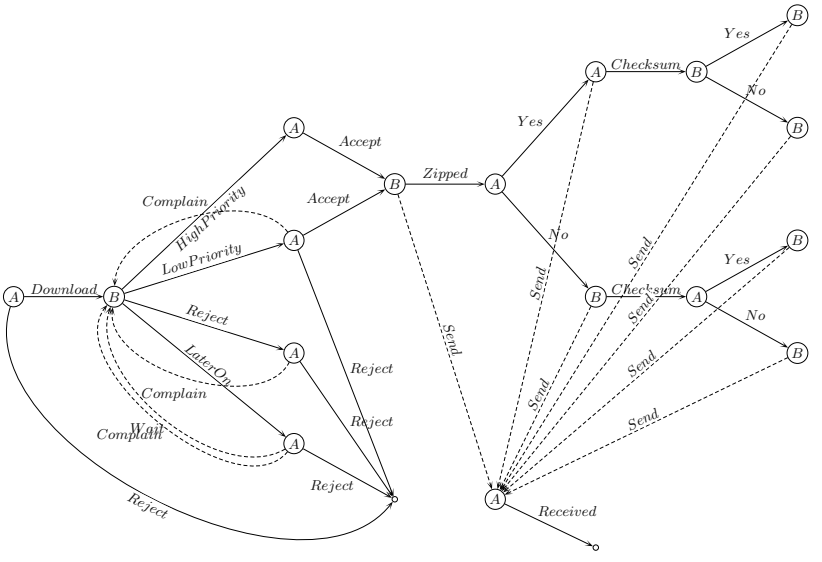

Fig. 1. Example of peer to peer system

For instance, after agents agree to transmit a file, the sender can ask the other agent whether it wants the file to be compressed with a *zip* compressor before sending it. Moreover, it can ask the receiver whether it wants the sender to check the integrity of the file before sending it (i.e., whether it can be correctly open with its respective reader application). In both cases, the sender will ask the receiver only if it has the suitable applications to perform them.

Let us comment the details of the protocol. As no dialog has been compressed yet, all represented messages are atomic. States of the requester and of the provider are marked by letters A and B, respectively. Firstly, the requester asks the provider for sharing and sending the file. Then, the provider can accept to send it with high priority, accept to send it with low priority, refuse the sharing, or ask the requester for waiting for a while and ask again. In the first two cases, the requester can accept. In the last case, the requester can wait for a while and ask again. In the last three cases, the requester can complain and ask for a better deal. Finally, in all cases the requester can abort the operation. Let us suppose that both partners accept to share and send the file. Then, if the provider has a zip compression application, it asks the requester whether it wants the file to be compressed. After, if the provider has the corresponding application to check the integrity of the file, it asks the requester whether it wants that application to be used before the sending. After this operation (or in a previous point if the needed application was not available) the provider begins the sending of the file, and finally the requester acknowledges the reception of it. Later on we will recall this example to present how dialogs are compressed. □

Let us note that any communication a partner i can perform from a state $s \in S_i$ must be distinguishable from any other communication it can perform from s; otherwise, the receiver of the message would be unable to properly interpret it. Without loss of generality, let us assume that we use the simplest signature to code our messages, that is, the binary signature. The binary code used to denote a message in each state should be different to any other binary code representing a different message in that state. Moreover, we must require that the binary code associated to a transition is not a *prefix* of another binary code associated to another transition of the same state. Let us suppose that c and c' are codifications of the messages m and m' respectively, and let us also suppose that c is a prefix of c'. Then, when a partner receives the codification c it cannot know whether the message m has been completely sent, or whether the message m' is still being sent. Let us include explicitly the binary codes associated to transitions in our formalization. Given a `DFSM` we construct a derived *codified* `DFSM` by assigning binary codes to each message/transition.

Definition 2. We say that a pair $C = (D, \eta)$ is a *Codified Dialog Finite State Machine*, in short `CDFSM`, if $D = (S_1, S_2, M_1, M_2, T_1, T_2)$ is a `DFSM` and the function $\eta : T_1 \cup T_2 \longrightarrow \{0,1\}^*$ assigns a binary code to every transition in D such that for any state $s \in S_i$ there do not exist two transitions $t_1, t_2 \in \texttt{Tran}(s)$ fulfilling that $\eta(t_1)$ is a prefix of $\eta(t_2)$. □

Let us note that any code is a prefix of itself. Hence, equal codes in different transitions of the same state are forbidden according to the previous definition.

In order to reduce the length of communications, it is desirable that codes associated to messages are as shorter as possible, while keeping their distinguishing power. In any codification, binary codes associated to different transitions from the same state might have different lengths. Obviously, it is preferable that most frequent messages are associated to shorter codes. Next we introduce a notion to denote a `CDFSM` where the codification of transitions in all states is *optimal*. A locally minimal codification is a `CDFSM` where the weighted average length of the messages is minimal. That is, when the length of each message is multiplied by its probability, the average of these products is minimal.

Definition 3. Let $D = (S_1, S_2, M_1, M_2, T_1, T_2)$ be a `DFSM`. A *locally minimal codification* of D is a function $\eta : T_1 \cup T_2 \longrightarrow \{0,1\}^*$ such that (D, η) is a `CDFSM` and there does not exist another function θ such that (D, θ) is also a `CDFSM` and there is a state $s \in S_1 \cup S_2$ with

$$\begin{array}{rl} & \sum\{\texttt{length}(\theta(t)) \cdot p \mid t = (s, m, p, s') \in \texttt{Tran}(s)\} \\ < & \sum\{\texttt{length}(\eta(t)) \cdot p \mid t = (s, m, p, s') \in \texttt{Tran}(s)\} \end{array}$$

If η is a locally minimal codification of D then we say that $C = (D, \eta)$ is a *locally minimal* `CDFSM`. □

A locally minimal `CDFSM` represents an optimal codification when we deal with compression *at the message level*, that is, when compression is applied only to atomic messages. Let us suppose that we construct a `DFSM` where all messages

are atomic, and we obtain its locally minimal CDFSM. If we constrain ourselves to perform a compression at the message level, then such a CDFSM provides a codification where the average length of any dialog between partners is minimal. However, as we will show later, our compression strategy will go one step further by introducing compound messages and compressing whole dialogs. Let us note that after the suitable compound messages are found and inserted into our DFSM, the best codification for our compression *at the dialog level* is provided by the corresponding locally minimal CDFSM as well.

The codification of the available messages in a state is independent from any other codification in other state. Thus, the problem of finding the locally minimal codification in a DFSM can be decomposed into the simpler problem of finding the optimal codification for each state. Let us define how to codify a set of pairs *(message,probability)* representing the messages of a state. This codification will be applied to each state of a DFSM to create its corresponding CDFSM.

Definition 4. Let $R = \{r_1, \ldots, r_n\}$ be a set of pairs $r_i = (m_i, p_i)$ where m_i is a message and $0 < p_i \leq 1$ is a probability. A *codification function for* R is a function C which returns for every message m_i a binary codification $\alpha \in \{0,1\}^*$ such that there do not exist $(m_j, p_j), (m_k, p_k) \in R$ such that $C(m_j)$ is a prefix of $C(m_k)$. We say that C is a *minimal codification function* if there does not exist another codification function B for S with $\sum\{\texttt{length}(B(m_i)) \cdot p_i \mid (m_i, p_i) \in S\} < \sum\{\texttt{length}(C(m_i)) \cdot p_i \mid (m_i, p_i) \in S\}$. □

As it is well known, the Huffman algorithm [4] computes the minimal codification function associated to a set of *symbols* (in our context, messages) and *frequencies* (in our context, probabilities). Thus, if we apply it to each state, we trivially obtain the locally minimal codification of a DFSM.

Lemma 1. Let $D = (S_1, S_2, M_1, M_2, T_1, T_2)$ be a DFSM. Let η be a function $\eta : T_1 \cup T_2 \longrightarrow \{0,1\}^*$ such that for every transition $t = (s, m, p, s') \in T_1 \cup T_2$ we have $\eta(t) = C_s(m)$, where C_s is a minimal codification function of $\{(m, p) \mid (s, m, p, s') \in \texttt{Tran}(s)\}$. Then, η is a *locally minimal codification of dialog* D. □

Let us remind that atomic messages transmit a single message while compound messages can represent proposals, refusals or acceptance of dialogs. Thus, to identify the actual messages communicated in a sequence of transitions, each message must be analyzed. Let us define the *real* meaning of a concatenation of messages (either atomic or compounds). For the sake of clarity, in the next definition we represent the sequences of messages as lists.

Definition 5. We define the *effective messages* belonging to a given list of messages by the following recursive expression:

$$\begin{aligned}
\texttt{effec}([\,]) &= \varepsilon \\
\texttt{effec}([(m_1 \cdots m_n?) \mid S]) &= \texttt{effec}(S) \\
\texttt{effec}([(m_1 \cdots m_n!) \mid S]) &= [m_1 \cdots m_n \mid \texttt{effec}(S)] \\
\texttt{effec}([(m_1 \cdots m_k!m') \mid S]) &= [m_1 \cdots m_k m' \mid \texttt{effec}(S)] \\
\texttt{effec}([(m_1 \cdots m_n\#) \mid S]) &= [m_1 \mid \texttt{effec}(S)] \\
\texttt{effec}([m \mid S]) &= [m|\texttt{effec}(S)]
\end{aligned}$$

□

Let us remark that a compound message as $m_1 \cdots m_n?$ does not represent any effective message, as it is necessary to wait until the other partner decides whether to accept the proposal or not. Hence, such messages do not add any effective message to the list.

Let us now consider how the current state of a DFSM evolves after a given sequence of transitions. Sequences of messages will be denoted by *evolutions*. Evolutions will be labelled by the sequence of messages labelling each transition and the probability of performing the whole sequence. That probability will be computed by multiplying the probabilities of each transition involved in the evolution. Let us remark that loops of states may appear in evolutions. However, it is not possible to have loops of length 1 since this would mean that a partner can send two messages before the other one sends anything.

Definition 6. Let $D = (S_1, S_2, M_1, M_2, T_1, T_2)$ be a DFSM. Besides, let $t_1 = (s_1, m_1, p_1, s_2)$, $\ldots$, $t_{n-1} = (s_{n-1}, m_{n-1}, p_{n-1}, s_n)$ be transitions such that for any $1 \leq i \leq n-1$, if $odd(i)$ then $t_i \in T_j$ and otherwise $t_i \in T_{3-j}$ for some $j \in \{1, 2\}$. Then, we write $\sigma = s_1 \xrightarrow{\texttt{effec}([m_1 \cdots m_{n-1}])}_{\pi} s_n$, where $\pi = \prod p_i$, we say that σ is an *evolution* of D. We denote the set of evolutions of D by $\texttt{Evol}(D)$.

Let $C = (D, \eta)$ be a CDFSM. Then, the *code length* of the evolution σ, denoted by $\texttt{clength}(C, \sigma)$, is defined as $\sum_{1 \leq i \leq n} \texttt{length}(\eta(t_i))$. □

3 Introducing Shortcuts

A *shortcut* is a sequence of states and transitions that are added to the original machine to reduce the codification length of a frequent evolution[3]. The aim of a shortcut is to allow communicating partners to perform a given sequence of messages with only a pair of messages (a proposal and an acceptance). In order to do it, new compound messages are inserted to represent the whole sequence. Obviously, reducing the codification length of a single evolution may produce a penalty in the codification lengths of other evolutions. Actually, the addition of available messages in some state requires, in general, to enlarge the codes of the messages to properly distinguish them. Nevertheless, if shortcuts are carefully introduced, the overall weighted average length of the evolutions will be reduced.

Next we explain the basic ideas underlying the construction of a shortcut, ant then we present them more formally in Algorithm 1. Let us suppose that there is an evolution from a state s_1 to a state s_n labelled by messages $a_1, a_2, \ldots, a_{n-1}$ in a DFSM, and we wish to shortcut it. This evolution will also appear in the modified DFSM. In addition, two new types of evolutions are added:

– First, a new evolution of two-steps is introduced. In s_1, the first partner proposes the whole compound evolution and then the second partner can accept it, reaching the state s_n in only two steps. These two new transitions

[3] The length of this evolution must be even. The reason is that the second partner is the one finishing the shortcut. Thus, the next message after the shortcut sequence should be sent by the first partner

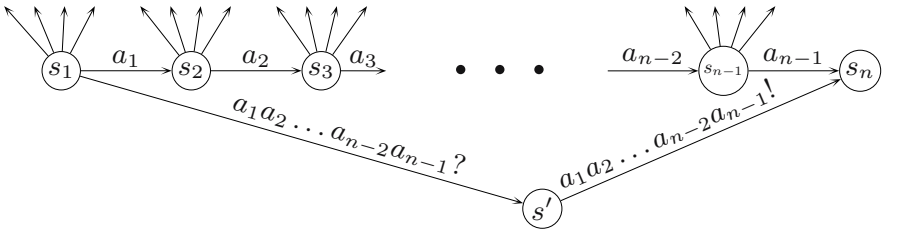

Fig. 2. First approach for shortcutting $a_1a_2\ldots a_n$ from state s_1 to state s_n

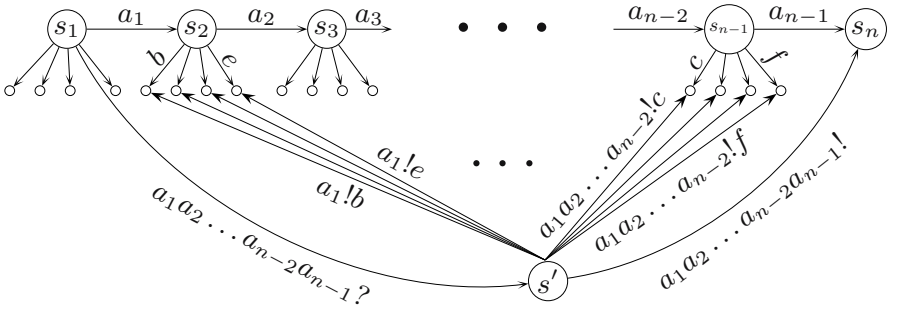

Fig. 3. First approach for including refusals in a shortcut

are labelled by $a_1a_2\cdots a_{n-1}?$ and $a_1a_2\cdots a_{n-1}!$, respectively (see Figure 2). Alternatively, after the initial compound proposal, the second partner can decide not to accept it. In such a case, we must indicate both the accepted part of the proposal and the first message b sent after that part. Let us suppose that the first i messages of the shortcut are accepted, but the $(i+1)$-th is not. The final state after the refusal will not be the $(i+1)$-th state of the original evolution. In fact, the final state must be the state reached after the $(i+1)$-th state when a message b is produced (see Figure 3). However, some information is missing: When the first partner does *not* want to perform the whole dialog, we must recompute the rest of probabilities of the original path, as these probabilities are *conditioned* to that fact.

– The second step of the construction comes from that case, that is, the case when the first partner is interested in sending the first message of the considered compound message but not the whole compound message. For dealing with such a case, we need to *clone* all the intermediate transitions and states of the original evolution (see Figure 4). This new cloned evolution will be as the original regarding actions, but the probabilities will be updated with the new extra information: We know that the first partner does not desire to perform the shortcut dialog. Moreover, the last transition of the evolution, denoting the willingness to perform the whole dialog, is not cloned.

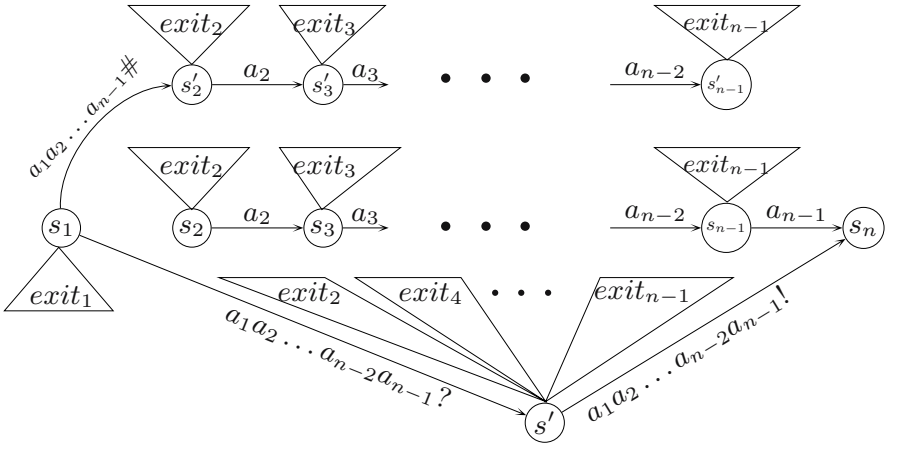

Fig. 4. A shortcut for $a_1a_2\ldots a_{n-1}$ from state s_1 to state s_n

Algorithm 1. Let $D = (S_1, S_2, M_1, M_2, T_1, T_2)$ be a DFSM and let us consider a sequence of transitions $s_1 \xrightarrow{m_1}_{p_1} s_2 \xrightarrow{m_2}_{p_2} \cdots \xrightarrow{m_{n-1}}_{p_{n-1}} s_n$. Let $\sigma = s_1 \overset{\rho}{\Longrightarrow}_{\pi} s_n$ with $\pi = \prod_{i=1}^{n-1} p_i$ and $\rho = \texttt{effec}(m_1 \cdots m_{n-1})$. Let $s_1 \in S_k$ and let $l = 3 - k$. A *shortcut* for σ in D is a new DFSM $D' = (S'_1, S'_2, M'_1, M'_2, T'_1, T'_2)$ constructed as follows:

- $S'_k = S_k \cup A_k$, where $A_k = \{s'_i \mid 1 \le i \le n-1 \ \wedge \ odd(i)\}$ is a set of fresh states. Intuitively, this step represents the inclusion of the odd states of the *cloned* evolution.
- $S'_l = S_l \cup A_l$, where $A_l = \{s'_i \mid 1 \le i \le n-1 \wedge even(i)\} \cup \{s'\}$ is a set of fresh states. Intuitively, this step represents the inclusion of the even states of the *cloned* evolution plus the accepting/rejecting state s' of the whole dialog.
- $M'_k = M_k \cup \{(\rho?), (\rho\#)\}$. Intuitively, we add a proposal to perform the shortcut as well as a message to perform the first step of the shortcut but not the whole one (i.e., going to the cloned evolution, see Figure 4).
- $M'_l = M_l\{(\rho!)\} \cup \bigcup_{1 \le i \le n-1, even(i)} \{(\texttt{effec}(m_1 \cdots m_{i-1}!m)) \mid \exists p, t : s_i \xrightarrow{m}_p t\}$. Here we add the acceptance and rejections (that is, acceptances up to a certain point) of the proposal.

Besides, transitions sets T'_k and T'_l are constructed as follows, where $\pi'_0 = 1$, $\pi_i = \prod\{p_k \mid odd(k) \ \wedge \ i \le k \le n-1\}$ and $\pi'_i = \prod\{p_k \mid even(k) \ \wedge \ 1 \le k \le i\}$:

- $T'_k = (T_k \setminus \{s_1 \xrightarrow{m_1}_{p_1} s_2\})$
  $\cup \{s'_i \xrightarrow{m_i}_{p-\pi_i} s'_{i+1} \mid \ s_i \xrightarrow{m_i}_p s_{i+1} \ \wedge \ odd(i) \ \wedge \ 3 \le i \le n-2\}$
  $\cup \{s_1 \xrightarrow{(\rho?)}_{\pi_1} s'\} \cup \{s_1 \xrightarrow{(\rho\#)}_{p-\pi_1} s'_2\}$
  $\cup \{s'_i \xrightarrow{m}_{p+p\cdot\pi_i} u \mid s_i \xrightarrow{m}_p u \ \wedge \ u \ne s_{n+1} \ \wedge \ odd(i) \ \wedge \ 1 \le i \le n-1\}$
  After removing the first transition of the original evolution, we have to add the odd steps of the cloned evolution. The probabilities change because we

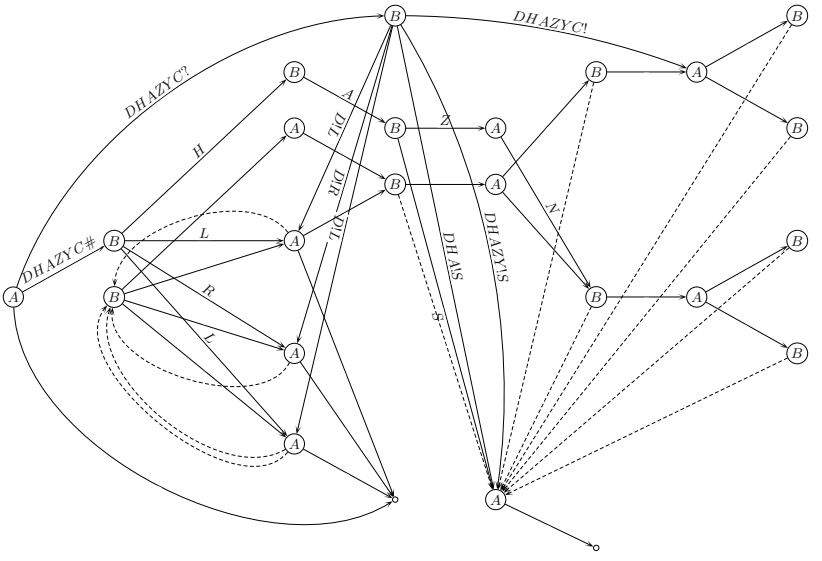

Fig. 5. Example of shortcut in the peer to peer system

know that the whole shortcut was not proposed. Additionally, we add the shortcut proposal, the first step of the cloned evolution, and the transitions to leave the cloned evolution.

- $T'_l = T_l \cup \{s'_i \xrightarrow{m_i}_p s'_{i+1} \mid s_i \xrightarrow{m_i}_p s_{i+1} \ \wedge \ even(i) \ \wedge \ 2 \leq i \leq n-2\}$
  $\cup \{s'_i \xrightarrow{m}_p u \mid s_i \xrightarrow{m}_p u \ \wedge \ u \neq s_{n+1} \ \wedge \ even(i) \ \wedge \ 1 \leq i \leq n-1\}$
  $\cup \{s' \xrightarrow{(\rho!)}_{\pi'_r} s_n \mid r = max\{i \mid 1 \leq i \leq n-1 \ \wedge \ even(i)\}\}$
  $\cup \{s' \xrightarrow{(\texttt{effec}(m_1 \cdots m_{i-1} ! m))}_{p \cdot \pi'_{i-2}} u \mid s_i \xrightarrow{m}_p u \ \wedge \ u \neq s_{n+1} \ \wedge \ even(i)$
  $\wedge \ 1 \leq i \leq n-1\}$

  First we add the even steps of the cloned evolution. Note that in this case probabilities do not change because the second partner is not the one rejecting the shortcut. Then, we also add the corresponding transitions to leave the cloned evolution. Finally, we add both the acceptance of the shortcut proposal and the set of rejections to it. For the rejections, we specify which part was accepted and the probabilities are changed accordingly. □

Example 2. Let us consider that a shortcut is introduced in the system described in Example 1. Let us suppose that two agents A and B use to share with each other every file they have, always with high priority. Besides, most of the times they want the file to be compressed, but they do not want the integrity of the file to be checked. Let us suppose that this sequence of acts is performed in 95 per cent of times they negotiate. Then, it is clear that the whole trace of transitions performing all of this acts (from the initial request up to the sending of the file) is a good candidate to perform a shortcut. Let σ denote this whole

shortcut from the first state. Let us note that, in spite that the first message (the simple request to download) will have to be denoted with a larger code of bits, the average length of symbols will decrease, because symbol σ will be performed most of the times, while the original first transition will be so rare (actually, the original request transition will be performed *only* if the requester does not want to perform σ). As a result, a lot of communications will be saved. Obviously, after the requester proposes σ, the provider can refuse it (though it will also be rare). In this case, depending on the part of σ the provider accepts, it will be conducted to a different state. Besides, if the requester wants to download a file but it does not want to perform σ, the `DFSM` is conducted to the sequence of *cloned* states. Each cloned state is related to the corresponding state of the original path. The resulting `DFSM` after the inclusion of the shortcut is depicted in Figure 5. For the sake of clarity, only those transitions directly related to the new states are included. Besides, messages are referred by their initials. □

4 Conclusions and Future Work

In this paper we have presented a method to reduce the communications in communicating systems and network protocols. By *shortcutting* frequent dialogs, the number of messages needed for performing them is notably reduced. Moreover, not only the number of messages, but also the *size* of them can be reduced (on average) by applying our strategy.

We think that this methodology can be applied to the design of new protocols. However, further research must be done to make our framework completely practical. Besides, the reciprocal influence of shortcuts must be studied. We have provided a method to decide whether we should introduce a single shortcut in a `DFSM`, but our framework allows to introduce several shortcuts consecutively in the same `DFSM`.

References

1. B.S. Bosik and M.U. Uyar. Finite state machine based formal methods in protocol conformance testing. *Computer Networks & ISDN Systems*, 22:7–33, 1991.
2. G.J. Holzmann. *Design and Validation of Protocols*. Prentice Hall, 1990.
3. J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 1979.
4. D.A. Huffman. A method for the construction of minimum redundancy codes. *Proceedings of the Institute of Radio Engineers*, 40(9):1098–1101, 1952.
5. R.P. Kurshan. *Computer-aided Verification of Coordinating Processes*. Princeton University Press, 1995.
6. T.A. Welch. A technique for high-performance data compression. *IEEE Computer*, 17(6):8–19, 1984.
7. J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.