Timeout Estimation using a Simulation Model for Non-repudiation Protocols

Mildrey Carbonell, Jose A. Onieva, Javier Lopez, Deborah Galpert
Computer Science Department, E.T.S. Ingeniería Informática University of Malaga, Spain
{mildrey,onieva,jlm,galpert}@lcc.uma.es

Jianying Zhou
Institute for Infocomm Research, Singapore
jyzhou@i2r.a-star.edu.sg

Abstract. An essential issue for the best operation of non-repudiation protocols is to figure out their timeouts. In this paper, we propose a simulation model for this purpose since timeouts depend on specific scenario features such as network speed, TTP characteristics, number of originators and recipients, etc. Based on a one-to-many Markowicth's protocol simulation model as a specific example, we have worked out various simulation experiments.

1 Introduction

Non-repudiation is a security service that is essential for many Internet applications, especially for e-commerce, where disputes between customers and merchants should be solved using digital evidences. Non-repudiation service must ensure that no party involved in a protocol can deny having participated in part or in the whole of it. An important requirement is *fairness* with which neither party can gain advantage by quitting prematurely or otherwise misbehaving during the protocol. .

Most of the non-repudiation solutions have been defined by means of a protocol using a *Trusted Third Party* (TTP) that plays the role of an intermediary between the participating entities. This entity participates in each step of the protocol may cause a communication bottleneck. Nevertheless, Zhou and Gollmann presented a protocol [1] where the TTP intervenes during each execution as a "low weight notary" rather than as an intermediary.

Some works on multi-party scenarios have been developed in similar topics, such as fair exchange [2,3]. The first effort to generalize non-repudiation protocols was presented by Markowitch and Kremer in [4,5] to allow one originator to send the same message to multiple recipients using a single key. An extension of the latter was presented in [6] where the originators could send different messages with a single key. In [7], an intermediary non-repudiation multi-party protocol was developed.

So far, most of the non-repudiation protocols (two-party or multi-party scenario) include diverse timeouts in their specifications. We have no reference about any proposed values or a procedure to estimate those timeouts. Due to the fact that these timeouts depend on real system conditions (e.g., network, involved parties, TTP capacity etc.), we are proposing the use of a simulation model in order to estimate approximated values of the timeout variable.

In this paper, we try to demonstrate, by means of a multi-party scenario example, how event-oriented simulation can be considered as a tool to estimate those timeouts, which can be adapted to the real conditions of each implementation. We select Kremer-Markowitch protocol, presented in section 2, because it is the first multi-party extension and its events are similar to the protocol in [6]. In section 3, we describe the event-oriented simulation model specifications and entities. The main events are shown in section 4. Finally, in section 5, we give different examples with this simulation model.

2 Kremer-Markowitch Protocol

In this section, we introduce an extension of the Zhou-Gollmann protocol performed by Kremer and Markowitch [4]. This extension uses the same key *k* for each recipient R_i, such that, an encrypted message *c*, evidence of origin (*EOO*), evidence of submission (*Sub*) and evidence of confirmation (*Con*) are generated for each protocol run. To ensure the fairness, the key is only revealed to those recipients *R*' that replied with evidence of receipt. This is achieved with a public-key group encryption scheme [8]. Some useful notation in the protocol description is as follows.

- $S_A(X)$: digital signature of user *A* over message *X*
- $E_K(X)$: encryption of message *X* with key *K*
- $h(X)$: hash function
- $\leftrightarrow$: fetch operation
- $A \Rightarrow \Pi$: multicast from entity *A* to the set Π
- *O :* originator
- *R* : set of intended recipients
- *R'*: subset of *R* that replied to *O* with evidence of receipt
- *M* : message being sent from *O* to *R*
- *k* : key being selected by *O*
- $c = E_k(M)$: message encrypted with *k*
- $l = h(M, k)$: label of message *M* and key *k*
- *t* : a timeout chosen by *O*, before which the TTP has to publish some information
- $E_{R'}(k)$: a group encryption scheme that encrypts *k* for the group *R'*
- $EOO = S_O(feoo, R, l, t, c)$: evidence of origin
- $EOR_i = S_{Ri}(feor, O, l, t, c)$: evidence of receipt of each R_i
- $Sub_k = S_O(fsub, R', l, t, E_{R'}(k))$: evidence of submission of *k* to the TTP
- $Con_k = S_{TTP}(fcon, O, R', l, t, E_{R'}(k))$: evidence of confirmation of *k* by the TTP.

The protocol is as follows.

1. $O \Rightarrow R$: R, l, t, c, EOO
2. $R_i \rightarrow O$: O, R_i, l, EOR_i where $R_i \in R$

$3.\ O \rightarrow TTP: \quad R', l, t, E_{R'}(k), Sub_k$
$4.\ R'_i \leftrightarrow TTP: \quad O, R', l, E_{R'}(k), Con_k \qquad$ where $R'_i \in R'$
$5.\ O \leftrightarrow TTP: \quad O, R', l, E_{R'}(k), Con_k$

The originator *O* multicasts to all recipients *R* the evidence of origin corresponding to the encrypted message *c* in step 1. Then, some recipients R_i (or all of them) send evidence of receipt EOR_i in *step 2*. In the next step, *O* sends *k* and evidence of submission Sub_k to the TTP in order to obtain evidence of confirmation Con_k in step 5. As we assume that the communication channel between *O* and the TTP is not permanently broken, *O* will be eventually able to send *k* and Sub_k to the TTP in exchange for Con_k at any time before timeout *t*.

In step 4, each recipient R'_i fetches $E_{R'}(k)$ and Con_k from the TTP at any time before *t* and stores it together with *EOO* as evidence to prove that message *m* was originated and sent by *O*; and the latter fetches Con_k from the TTP and stores it as evidence to prove that *k* is available to *R'*.

Timeout *t* constitutes one of the halt conditions in Kremer-Markowicth protocol due to the fact that the TTP cannot publish the cipher key if it receives *k* some time *t'* after the timeout. On the other hand, timeout *t* can also be used to stop originator's key publication request sent to the TTP if the latter could not be connected before deadline *t*. Besides, the recipients should halt the protocol if the key has not been published after time *t*. Both halt conditions for *O* and *R* would avoid useless loops.

The estimation of this timeout *t* depends on the real features and conditions of the implemented scenario including number of originators and recipients, TTP capacity and the network speed. In the next section we present the simulation model of the protocol described in this section.

3 Simulation Model

The following model is useful in order to estimate the timeout and to diagnose some possible problems in the implementation of the protocol, starting from different values of the critical system variables such as: low connection speed, shortage of TTP storage capacity and delay in the messages due to firewall protection or other security schemes. This diagnosis could be used to model further improvements in the real scenario to find better implementations, shortest waiting time and adequate TTP features. The model includes 15 different events:

- Originators send messages to recipients `(event 1: Message generation, event 2: Message arrival to R)`.
- These originators will wait for *EOR* and then send a key publication request to the TTP `(event 3: `*`EOR`*` arrival to O)`. In the simulation model, O will wait for all *EOR* in order to estimate the delaying time when all *R* send *EOR* to O in a real execution.
- The TTP publishes the key if it has enough connection and storage capacity `(event 4: Arrival of the publication request to the TTP, event 6: Disconnection of O's publication request)`.

– Otherwise, O should retry the request later (`event 5: O's key publication request retry`).
– Once the key is published, the originator and the recipients can start *Con* requests (`event 7: O's` *Con* `request, event 8: R's` *Con* `request`).
– If allowed by FTP resources, the TTP opens a connection with the involved entity, verifies the key of the message and outputs an affirmative or negative response to the request (`event 9: Connection for O's` *Con* `request, event 10: Connection for R's` *Con* `request, event 14: O FTP disconnection, event 15: R FTP disconnection`).
– If FTP resources are exhausted, the involved entity should retry the connection later (`event 11: O's` *Con* `request retry, event 12: R's` *Con* `request retry`).
– The key is maintained in the TTP's database until timeout *t* (`event 13: Key deletion on the TTP`).
– When all involved entities have verified the key, one protocol execution would have finished.

In the real scenario, the TTP needs to process many protocol executions with similar or different originators. We could imagine an electronic bookshop, during the whole day selling books (and thus using a non-repudiation protocol). In this paper, the stop criteria will be the end time of the simulation event.

Problem: Estimate the timeout t that O sends to R in the first step of the protocol according to the real scenario.

Goals: To reduce the delay in the entire system while guaranteeing a complete execution of the protocol steps, we need to find the influence of modifications on:

- number of originators and recipients
- number of messages that the originators send to the recipients
- network speed
- capacity of connection to publish in the TTP
- FTP capacity of connection to the TTP
- storage capacity in the TTP
- key publication time in the TTP
- time between successive retries of connections

We can model the protocol with event-oriented simulation [9] due to the fact that the generation and the reception of messages are asynchronous processes that evolve a finite number of events. Following we present the entities of the simulation model and its variables.

Table 1: Simulator entity

Entity 1: Simulator (S)	
Variables	**Description**
Input variables	

FinalTime	Final simulation time
Recipients	Number of Recipients (R)
Originators	Number of originators (O)
MsgGenDist	List of message generation distributions for each O (step 1)
CommunicationOR	Matrix of delay distributions of network messages between O and R (step 1)
CommunicationOTTP	List of delay distributions of network messages between O and the TTP (step 3)
CommunicationRTTP	List of delay distributions of network messages between R and the TTP (step 4)
EORsendDist	Delay distribution of the *EOR* message (step 2)
PUBConnectionDist	Time distribution of O's connection to publish the key in the TTP (step 3)
FTPConnectionDist	FTP connection time distribution of O and R (steps 4 and 5)
State variables	
CurrentTime	Current simulation time
Lentity	List of entities
Levent	List of events

Table 2: Message entity

Entity 2: Message (M): This entity is created by originators. Each originator is able to create many messages.	
Variables	**Description**
State variables	
IdM	Unique identifier of message *m*
CreationTime	Creation time (step 1)
State	States of the message: St1 : It is being sent to R (step 1) St2: O is waiting for all *EOR* (step 2) St3: O is trying to publish the key in the TTP (step 3) St4: The key has been published in the TTP (step 3) St5: The key was deleted from the TTP
Nbr_EOR	Number of R that sent *EOR* (step 2)
Output variables	
WaitRTime	Total waiting time for all *EOR* (step 2)
PubDelayTime	Key publication delay time (step 3)
Nbr_PUBRetries	Number of O's key publication request retries (step 3)

Table 3: Originator entity

Entity 3: ORIGINATOR (O)	
Variables	**Description**
Input variables	

Time_btw_PUBRetries	Time between successive retries of O's key publication requests (step 3)
Time_btw_FTPRetries	Time between successive retries of O's *Con* requests (step 5)
State variables	
IdO	Originator's unique identifier
LMsg	List of messages generated by O (step 1)
Nbr_Msg	Number of messages generated by O (step 1)
Output variables	
Nbr_PublicMsg	Number of published keys (step 3)

Table 4: Recipient entity

Entity 4 : RECIPIENT (R)	
Variables	**Description**
Input variables	
Time_btw_FTPRetries	Time between successive retries of R's *Con* requests (step 4)
State variables	
IdR	Recipient's unique identifier
LReceivedMsg	List of received messages (step 2)
Output variables	
Nbr_ReceivedMsg	Number of received messages (step 2)

Table 5: TTP entity

Entity 5: TTP	
Variables	**Description**
Input variables	
Max_StorageKTime	Key storage time in the TTP
CapacPUBConnection	Publication connection capacity
CapacFTPConnection	FTP connection capacity
CapacStorage	Storage capacity measured in number of keys
State variables	
Current_ConnectedPUB	Current number of publishing connected entities
Current_ConnectedFTP	Number of FTP connected entities
CapacOccupied	Occupied storage capacity
Output variables	
LPublicMsg	List of messages whose keys were published
Nbr_PUBMsg	Number of messages whose keys were published
Nbr_PUBRetries	Number of retries of O's key publication request caused by the lack of TTP connection capacity (step 3)
Nbr_PUBRetries_Str	Number of retries of O's key publication request caused by the lack of TTP storage capacity (step 3)

Nbr_O_Con_Retries	Number of retries of O's *Con* request (step 5)
Nbr_R_Con_Retries	Number of retries of R's *Con* request (step 4)
Nbr_Successful_O_Con	Total number of successful O's *Con* requests (step 5)
Nbr_Unsuccessful_O_Con	Total number of unsuccessful O's *Con* requests (step 5)
Nbr_Successful_R_Con	Total number of successful R's *Con* requests (step 4)
Nbr_Unsuccessful_R_Con	Total number of unsuccessful R's *Con* requests (step 4)

4 List of Main Model Simulation Events

Following, we describe the main publication key events (**1-6**). We can use *entity.variable* to refer to one variable of the entities (*S*, *M*, *O*, *R* and *TTP*).
For each event we describe the name and the input parameters inside the brackets.

```
Event 1: Message generation (O: originator)
Generate a message at time t=S.CurrentTime
    Increase O.Nbr_Msg
    M.IdM = O.IdO + O.Nbr_Msg
    M.CreationTime = S.CurrentTime
    M.State = St1
For i = 1 to S.Recipients do
    Add the event Message arrival to R (O,M,$R_i$) at time
        t=S.CurrentTime + Random value generated with
        S.CommunicationOR(O,Ri)
Add M to the list O.LMsg
Add the event Message generation (O) at time
    t=S.CurrentTime + Random value generated with
    S.MsgGenDist(O)
```

```
Event 2: Message arrival to R
(O: originator, M: message, R: recipient)
Add the message to the list R.LReceivedMsg
Increase the number of received messages R.Nbr_ReceivedMsg
Add the event EOR arrival to O (M,R) at time
    t=S.CurrentTime +
    Random value generated with S.CommunicationOR(O,R)
    + Random value generated with S.EORsendDist
```

```
Event 3: EOR arrival to O (M: message, R: recipient)
Increase M.Nbr_EOR
Change the state of the message M.State=St2
If M.Nbr_EOR = S.Recipients
    M.State=St3
    Update M.WaitRTime= S.CurrentTime - M.CreationTime
    Add the event **Arrival of the publication request to
        TTP (O, M, TTP)** at time
        $t=S.CurrentTime$ + Random value generated with
        $S.CommunicationOTTP(O)$
```

Event 4: Arrival of the publication request to TTP
(O: originator, M: message, TTP: trusted third party)

```
If $TTP.Current\_ConnectedPUB + 1 > TTP.CapacPUBConnection$
    Increase $TTP.Nbr\_PUBRetries$
    Add the event **O's key publication request retry
        (O,M)** at time
        $t = S.CurrentTime + O.Time\_btw\_PUBRetries$
Else
    If $TTP.CapacOccupied + 1 > TTP.CapacStorage$
        Increase $TTP.Nbr\_PUBRetries\_Str$
        Add the event **O's key publication request retry
            (O,M)** at time
            $t = S.CurrentTime + O.Time\_btw\_PUBRetries$
    Else
        Increase $TTP.Current\_ConnectedPUB$
        Add the event **Disconnection of O's publication
            request (O,M, TTP)** at time
            $t = S.CurrentTime$ + Random value generated
            with $S.PUBConnectionDist$
```

Event 5: O's key publication request retry
(O: originator, M: message)

```
Add the event **Arrival of the publication request to TTP
    (O, M)** at time
    $t = S.CurrentTime$ + Random value generated with
    $S.CommunicationOTTP(O)$
```

Event 6: Disconnection of O's publication request
(O: originator, M: message, TTP: trusted third party)

```
Update $M.PubDelayTime=S.CurrentTime - M.CreationTime$
Increase $O.Nbr\_PublicMsg$
Increase $TTP.Nbr\_PUBMsg$
Add the message to the list $TTP.LPublicMsg$
Increase $TTP.CapacOccupied$
Decrease $TTP.Current\_ConnectedPUB$
Change the state of the message $M.State=St4$
Add the event **O's Con request(M)** at time
    $t = S.CurrentTime$
Add the event **R's Con request(M)** at time
```

```
    t=S.CurrentTime for each recipient i.
  Add the event Key deletion in the TTP (TTP,M) at time
    t = S.CurrentTime + TTP.Max_StorageKTime
```

Main Program

```
Initialization of Simulator (S)
   -  Generate the events of Message Generation(O) for
      each O
   -  Add all entities to the simulator
   -  Initialize the input variables
While not empty S.LEvent and S.CurrentTime < S.FinalTime do
   -  E = The minimum time event in S.LEvent
   -  Delete E from S.LEvent
   -  S.CurrentTime = time of E
   -  Execute the procedure that handles the event
Do the report
   -  For each entity save the report
```

5 Output Analysis

We implemented an example of the described protocol in a 100Mbits network with 3000 machines. The originators send messages to the recipients with a uniform distribution between ½ hours and 1 hours (*S.MsgGenDist*). After one hundred executions of the protocol we calculated the following input distributions of the model:

- The network message delay distribution between originators and recipients, originators and the TTP, recipients and the TTP is a uniform distribution between 10ms and 17ms. (*S.CommunicationOR, S.CommunicationOTTP, S.CommunicationRTTP*)
- The delay distribution of the *EOR* reply is a uniform distribution between 15ms and 20ms. (*S.EORsendDist*)
- The time distribution of O´s connection to publish the key is an uniform distribution between 30ms and 50ms. (*S.PUBConnectionDist*)
- The FTP connection time distribution of the originators and the recipients is a uniform distribution between 25ms and 35ms. (*S. FTPConnectionDist*)

We estimated the key publication delay time *(M.PubDelayTime)* and the waiting time for all evidences of receipt (*M.WaitRTime*) with fixed initial conditions.

Notation

Input variables
- **NO** – Number of originators (*S.Originators)*

- **NR** – Number of recipients (*S.Recipients*)
- **C** – TTP storage capacity measured in number of keys (*TTP. CapacStorage*)
- **FTP** – FTP connection capacity (*TTP. CapacFTPConnection*)
- **TS** – Key storage time in the TTP (*TTP.Max_StorageKTime*)
- **RO** – Time between successive retries of O´s *Con* request
  (*O. Time_btw_FTPRetries*)
- **RR** – Time between successive retries of R´s *Con* request
  (*R. Time_btw_FTPRetries*)

Output variables

- **NM** – Number of generated messages in the experiment $\sum_{i=1}^{NO} Oi.Nbr\_Msg$
- **MP** – Number of messages whose keys were published on the TTP
  (*TTP.Nbr_PUBMsg*)
- **CPC** – Number of successive retries of O´s key publication request caused by the lack of TTP connection capacity (*TTP. Nbr_PUBRetries*)
- **CPA** – Number of successive retries of O´s key publication request caused by the lack of TTP storage capacity (*TTP.Nbr_PUBRetries_Str*)
- **CRO** – Number of successive retries of O´s *Con* request
  (*TTP.Nbr_O_Con_Retries*)
- **CRR** – Number of successive retries of R´s *Con* request
  (*TTP.Nbr_R_Con_Retries*)
- **SO** – Number of successful O´s *Con* requests (*TTP.Nbr_Successful_O_Con*)
- **SR** – Number of successful R´s *Con* requests (*TTP.Nbr_Successful_R_Con*)
- **UO** – Number of unsuccessful O´s *Con* requests (*TTP.Nbr_UnSuccessful_O_Con*)
- **UR** – Number of unsuccessful R´s *Con* requests (*TTP.Nbr_UnSuccessful_R_Con*)
- **ERT** – Average waiting time of all *EOR*

$$\frac{\sum_{i=1}^{NO}\left(\frac{\sum_{j=1}^{Oi.Nbr\_Msg} Mj.WaitRTime}{\text{Oi.Nbr\_Msg}}\right)}{NO}$$

- **PKT** – Average key publication delay time

$$\frac{\sum_{i=1}^{NO}\left(\frac{\sum_{j=1}^{Oi.Nbr\_Msg} Mj.PubDelayTime}{\text{Oi.Nbr\_Msg}}\right)}{NO}$$

Result:

	Input variables						
	NO	**NR**	**C**	**FTP**	**TS**	**RO**	**RR**
A	300	30	10500	9000	1min	20s	20s
B	5000	30	10500	9000	2min	20s	20s

C	10000	10	10500	9000	1min	20s	20s

Output variables										**Timeouts**	
NM	**MP**	**CPC**	**CPA**	**CRO**	**CRR**	**SO**	**SR**	**UO**	**UR**	**ERT**	**PKT**
4672	4669	0	0	0	0	4668	140041	0	0	10.75s	50.85s
76885	76833	0	0	0	0	76816	2304481	0	0	11.93s	51.97s
157850	157775	2000	0	0	0	157739	1577370	0	0	10.50	60.20s

The simulation estimation of the timeout ***t*** was:

- (**A**) 50.85s with 300 originators, 30 recipients. In this implementation of the protocol the originator would not wait more than 10.75s for the *EOR* in order to send the key publication request to the TTP.
- (**B**) 51.97s with 5000 originators, 30 recipients. The originator would not wait more than 11.93s for the *EOR* in order to send the key publication request to the TTP.
- (**C**) 60.20s with 10000 originators, 10 recipients. The originator would not wait more than 10.50s for the *EOR* in order to send the key publication request to the TTP.

An increase in the number of originators (example **C**) resulted in a slight increase in the **PKT**. The TTP need to publish more keys.

We can do others experiments with this simulation model like:

- The estimation of efficient initial conditions (**C, FTP, TS, RO, RR**) so that the protocol would operate without unsuccessful *Con* searches with a fixed number of originators and recipients. Obviously, these adjustments can help in the decision-making of a TTP investment process.
- The estimation of the larger number of originators combined with the fixed number of recipients and the fixed conditions in the TTP (storage and connection capacities).

In a future work we will develop those experiment.

The equipment used for the simulation was an Intel(R) Pentium(R) 4CPU, 1.60GHz, 224MB of RAM. The experiments proved the simulation model's effectiveness. The simulation model was implemented with Delphi 6.

6 Conclusion

An essential issue for the best operation of non-repudiation protocols is to figure out their timeouts. In this paper, we proposed a simulation model for this purpose since timeouts depend on specific scenario features such as network speed, TTP characteristics, number of originators and recipients, etc. This simulation would be very useful for a reliable and adequate implementation.

This simulation model could be extended to other security protocols in two-party and multi-party scenarios. In some future work, further simulation models could be carried out for more complex multi-party protocols like the intermediary non-repudiation protocol [7].

The model was proved with some experiments presented in this paper. We have not used a significant number of originators and recipients but now we are on the pursue of distributed simulation implementations to achieve the simulation of bigger scenarios with around 1000000 originators.

References

1. J. Zhou and D. Gollmann. "A fair non-repudiation protocol". Proceedings of 1996 IEEE Symposium on Research in Security and Privacy, pages 55-61, Oakland, CA, May 1996.
2. N. Gonzalez-Deleito and O. Markowitch. "An optimistic multi-party fair exchange protocol with reduced trust requirements". Proceedings of 4th International Conference on Information Security and Cryptology, pages 258–267, Seoul, Korea, December 2001.
3. J. Kim and J. Ryou. "Multi-party fair exchange protocol using ring architecture model". Proceedings of Japan-Korea Joint Workshop on Information Security and Cryptology, January 2000.
4. O. Markowitch and S. Kremer. "A multi-party non-repudiation protocol". Proceedings of 15th IFIP International Information Security Conference, pages 271-280, Beijing, China, August 2000.
5. O. Markowitch and S. Kremer. "A multi-party optimistic non-repudiation protocol". Proceedings of 3rd International Conference on Information Security and Cryptology, pages 109-122, Seoul, Korea, December, 2000.
6. J. Onieva, J. Zhou, M. Carbonell, and J. Lopez. "A multi-party non-repudiation protocol for exchange of different messages". Proceedings of 18th IFIP International Information Security Conference, Athens, Greece, May 2003.
7. J. Onieva, J. Zhou, M. Carbonell, and J. Lopez. "Intermediary non-repudiation protocols". Proceedings of IEEE Conference on Electronic Commerce, Newport Beach, CA, June 2003.
8. G . Chiou and W. Chen. "Secure broadcasting using the secure lock". IEEE Transaction on Software Engineering, Vol. 15, No. 8, August 1989.
9. J. Banks, J. Carson, and B. Nelson. "Discrete-event system simulation". Prentice Hall, 2000.