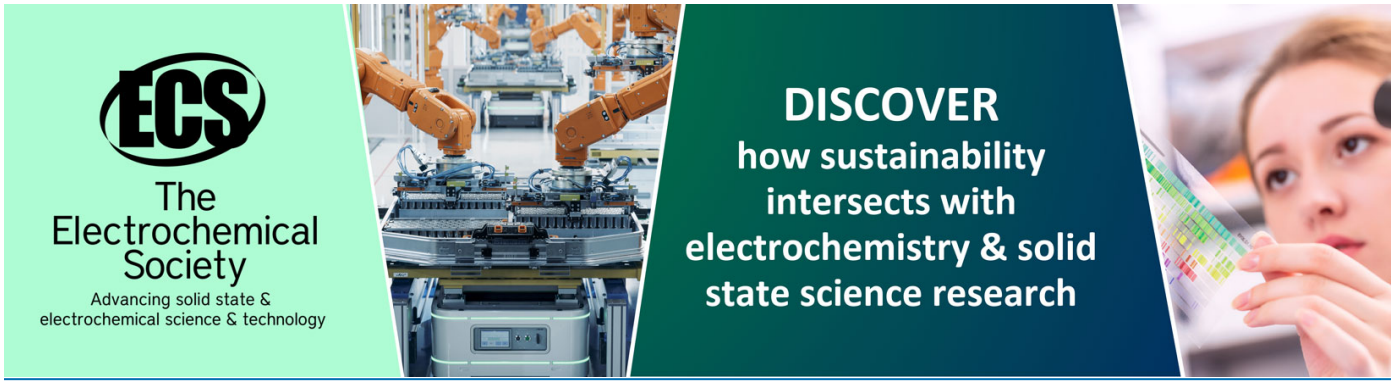
Directly executable formal models of middleware for MANET and Cloud Networking and Computing



Directly executable formal models of middleware for MANET and Cloud Networking and Computing

D. V. Pashchenko[1], Mustafa Sadeq Jaafar[2], S. A. Zinkin[3], D. A. Trokoz[4], T. U. Pashchenko[5] and M. P. Sinev[6]

[1] Professor, Penza State University, Penza, Russia

[2] Postgraduate, Penza State University, Penza, Russia

[3] Professor, Penza State University, Penza, Russia

[4] Assistant professor, Penza State University, Penza, Russia

[5] Assistant professor, Penza State University, Penza, Russia

[6] Assistant professor, Penza State University, Penza, Russia

E-mail: dmitry.pashchenko@gmail.com

Abstract. The article considers some "directly executable" formal models that are suitable for the specification of computing and networking in the cloud environment and other networks which are similar to wireless networks MANET. These models can be easily programmed and implemented on computer networks.

1. Introduction

Several new trends are opening up the era of Cloud Computing, which is an Internet-based development and use of computer technology. They are cheaper and more powerful processors, together with the "software as a service" (SaaS) and computing architecture, are transforming data centers into pools of computing service on a huge scale. Meanwhile, the increasing of networks bandwidth and their reliability yet flexible network connections make it even possible that clients can now subscribe some high quality services from data and software that reside solely on remote data centers or data pools. Although envisioned as a promising service platform for the Internet, the new data storage paradigm in the "Cloud" brings many challenging design issues which have profound influence on the security and performance of the overall system. One of the biggest concerns related to cloud data storage is that of data integrity verification at non trusted servers. What is more serious is that for saving money and storage space the service provider might neglect to keep whatever is necessary or delete rarely accessed data files which belong to an ordinary client. Consider the huge size of the outsourced electronic data and the client's constrained resource capability, the core of the problem can be generalized in the theme of how the client can find an efficient way to perform periodical integrity verifications without needing the local copy of data files. Considering the role of the verifier in the model where all the schemes presented before are falling into two categories: private auditability and public auditability. Although schemes with private auditability can achieve remarkably higher scheme efficiency, public auditability allows anyone, not just the client (data

owner) to challenge the cloud servers for the correctness of data storage while keeping no private information.

Then, clients are capable of handing over or to be more specific, to delegate the evaluation of the service performance to some independent third party auditors (TPAs), without the devotion of their computation resources. In the cloud, the clients themselves mostly are unreliable or may not be capable of affording the overhead of performing frequent integrity checks. Router free feature connecting to files on other computers and/or the Internet without the need of a wireless router is the main advantage of using an ad hoc network. So because of this, running an ad hoc network can be more affordable and easier to construct than a traditional network because you don't have the added cost of a router. However, if you only have one node for example only one computer then it won't be possible to construct an ad hoc network. Mobility - an Ad hoc network can be created on the fly in nearly any situation with multiple nodes (wireless devices) which they are existing for the purpose of connection establishment. For example: emergency situations in some rare and remote locations make a traditional network nearly impossible, but "For example the medical team in the army can easily utilize 802.11 radio NICs in their laptops and PDAs and other wireless devices and enable broadband wireless data communication as soon as they reach their destination. "The other feature is the Speed - creating an Ad hoc network from scratch is remarkably fast and only requires a few setting tunings and some changes and no additional hardware or software.

2. The results of previous works

Mobile ad hoc networks (MANET) consist of nodes that are able to communicate through the use of wireless mediums and form dynamic network topologies. The basic characteristic of these networks is the complete lack of any kind of infrastructure and the absence of any dedicated nodes that provide network management operations as the role of the traditional routers in fixed kinds of networks. In order to maintain the connectivity in a mobile ad hoc network all the nodes have to perform routing of network traffic. The cooperation of the nodes in an ad hoc network cannot be enforced by a centralized administration authority since one does not actually exist. Therefore, a network-layer protocol that is designed for such self-organized networks must enforce the connectivity and the security requirements in order to guarantee the undisrupted other operations of the higher layer protocols. Unfortunately most of the widely used ad hoc routing protocols have no high security considerations and trust all the participants (the nodes) to correctly forward routing and data traffic.

Proposed by one of the authors of this article in [1, 2, 3] modified Ack-Based scheme (Acknowledgment-Based) for node authentication with AODV (Ad hoc On-Demand Distance Vector) existing protocol in MANET. Ack-Based scheme also provides the possibility of detecting wormhole attacks and nodes misbehavior in ad hoc networks. Different researcher proposed different scheme for Ack-Based for providing security in mobile ad hoc network. But all the other proposed techniques suffered common problem that problem is the generation of huge amount of pack overhead and node ambiguity. Actually due to this problem the given scheme is not used in a general form. So we proposed the Ack-Based scheme with the help of finite state machine against the malicious nodes. The authors propose the 2Ack based scheme to detect the malicious links and to mitigate their malicious effects. This scheme is based on a 2Ack packet that is actually assigned a fixed route of two hops in the reversed/opposite direction of the received data between the sender and the receiver. In this scheme, each sender’s packet maintains the following parameters; (i) list of identifiers of data packets that have been sent out but haven’t been yet acknowledged, (ii) a counter of the forwarded data packets, (iii) a counter of the missed data packets. According to the value of the acknowledgements ratio (Rack), only a certain fraction of the sent data packets will be acknowledged in order to reduce the resulted overhead. This technique overcomes some of the weaknesses of the Watchdog/path rater such as: ambiguous collisions, receiver collisions and power control transmissions. Both of the previous presented works remain vulnerable to the attacks that are launched by groups of other malicious nodes. To encounter these attacks, must provide a framework to mitigate the damage caused by the colluding black hole attacks in AODV. The proposed technique actually has a moderate

overhead that is induced by the ACK which has been sent back by the destination node during selected intervals of data transfer periods. Throughout the data packets transmissions, a flow of special data packets is transmitted at random intervals of time along with the data. The reception of these special data packets invokes the destination to actually send out an ACK through different multiple routing paths. The ACK data packets take multiple routes to reduce the actual probability that all of the ACKs are getting dropped by the malicious nodes, and also to encounter for possible loss due to several broken routes or congestions in some certain nodes. If the source node does not get back any ACK packet, then it becomes aware of the presence of some potential attackers in the forwarding path. As a reaction to that, it broadcasts a list of suspected malicious nodes in order to isolate them from the network's routing table.

Without a fixed infrastructure and with the node mobility feature in ad hoc networks it is a great challenge as concerning the security issue. For security concerns various methods are used to enhance node authentication in mobile ad hoc networks. The authentication scheme of leader agent node and members of surveillance greatly reduces the relative calculating overheads and the communication costs. Generally, when leader agent node and surveillance nodes are not getting destroyed, the united nodes can effectively ensure the reliability and the authentication results will be more reliable. Suggestion for future work is to apply the modified Ack-Based FSA (Finite State Machine) scheme to other protocols to overcome the requirement of the memory, during the path discovery and path establishment it takes a little bit much time in comparison with normal Ack-Based schemes and also future minimized route calculation with finite state machine. Here in [1, 2, 3] proposed Modified ACK-Based scheme for node authentication with AODV existing protocol in MANET. ACK-Based scheme also provides the capability of detecting the wormhole attacks and node misbehaviors in ad hoc networks. Different researcher proposed different scheme for ACK-Based for providing security in mobile ad hoc network. But all these techniques suffered from a common problem which is the generation of huge amount of pack overhead and node ambiguity. Actually due to this problem the given scheme is not used in a general form. So we proposed the ACK-Based scheme with the help of the finite state machine concept for controlling the generation of data packets and also improve the performance of AODV protocol. Here first we will discuss basic ACK-Based scheme and then we will discuss the finite state machine and finally we will discuss the combined approach for ACK-Based scheme.

When a malicious node is receiving an application packet from a node and that packet is actually destined for some other node then instead of forwarding that packet, it simply just drops that application packet. This data loss may become severe when the number of malicious nodes present in network is becoming high. In the proposed work, we overcome this problem by identifying this kind of malicious behavior of nodes and then a route via such a node is never being chosen by its neighboring node to forward an application packet in the network.

When a node wants to send an application packet to another node which is not its immediate neighboring node then it sends a RREQ (request) packet to all of its neighboring nodes. If a neighbor knows a route to the destination of this packet then it sends a RREP (reply) packet that contains the next hop address to which the mentioned neighboring node will forward the packet to. Let us call this next hop address as the next to next hop address instead. The algorithm is described as follow:

1. The sender node forwards the application packet to one of its immediate neighbors delegating the responsibility of further forwarding it to that neighboring node which is the destined one. The sender also sets a timer (which is twice the network's diameter) to receive the acknowledgement from the destination node.
2. If an acknowledgement is received before the timer expires then the route is considered to be trusted/ valid and no further action is needed to be taken.
3. If the timer expires and an acknowledgement is not yet received then the route is not considered to be trusted/ valid. Now the sender sends an application packet to the next to next hop address node and again sets a timer to receive an acknowledgement.

- If the acknowledgement is received before the timer expires then the neighboring node is considered to be trusted and no further action will be needed, otherwise the neighboring node is considered as a half-trusted. In this situation, this half-trusted node will be under observation until it shows the same malicious behavior all over again when an application packet is forwarded to it the next time.

4. If the half-trusted node does not show any other malicious behavior when the application packet is forwarded to it the next time then it will be considered as a trusted-node, otherwise this half-trusted node will be considered as a malicious one and therefore the following actions will be performed:
   - No more RREP messages from this node will be considered.
   - The application packets will no longer be forwarded to this node.
   - New routes will be found and discovered in order to forward the application packets to those destinations that are having this un-trusted (malicious) node as the next hop address in the routing table.

3. Some "reconfigurable" directly executable formal models, suitable for the specification of computing and networking in the cloud environment and other networks similar to wireless networks MANET

The distributed programming computational infrastructures of Cloud and MANET networking and computing available globally for offering uniform services has become an important topic in Computer Sciences. The challenges are coming from the necessity of dealing at once with issues like the security, privacy, failures, communications, co-operation, mobility, resources usage, etc., in a setting where the demands and the guarantees can be very different for the many different components. This has stimulated research on abstractions and models that could provide the basis for the design and the analysis of network aware programs, where dynamically changing the logical and physical architecture of systems plays a crucial role. The range of applications of reconfigurable, or changeable, models proposed in this paper concerns the problem of global computing and communications, which is especially important for MANET applications and Cloud Computing. We will give some information on the used formalism. Logical and algebraic models describe a set of properties of network programs. However, they do not give an idea of why these properties are changing. Executable or operating models help us to monitor the behavior of the object. Thus, every executable formal model is an abstract program for some virtual machines. Some formal models can be executed immediately before the phase of the network programming. Such models we conventionally call as "the directly executable". Those models are classified as follows: the models of intermediating type, possessing properties of the logic-algebraic and the executable models.

The concept of "directly executable" formal models is a metaphor, which is used in the sense of Glushkov and his followers [10-17]. It means that the design of formal specifications is the last step for the programmer before the writing of the network programs for MANET and Cloud Computing.

First, for a unified view of a number of important directly executable models let us consider (dynamically changing, many-sorted) algebraic system:

$AS = (\boldsymbol{A}, P, F, I_{F0}, I_{P0}, Rules, \boldsymbol{M}, Z)$,

where $\boldsymbol{A} = \{A_1, A_2, \ldots, A_n\}$ – a finite set of some basic finite sets, or in another word "sorts" $A_1, A_2, \ldots, A_n$;

P – a finite set of predicate symbols;.

F – a finite set of function symbols;.

I_{F0} – initial interpretation of function symbols;

I_{P0} – initial interpretation of predicate symbols;

Rules – a finite set of modifications (or updates) to the interpretation of the predicate and function symbols;

$\boldsymbol{M}$ – a finite set of abstract modules;

Z: $\boldsymbol{M} \to \mathbf{P}(Rules)$ – a mapping from the set M to the set of subsets of *Rules*, where $\mathbf{P}$ is a Boolean symbol. Thus, as a result of the work of abstract modules of the set $\boldsymbol{M}$ dynamic algebraic system *AS* evolves, passing from one interpretations of predicate and function symbols to another.

We shall also use the known in algorithmic algebras operators-constants E (the identity operator, or "empty" operator) and N (nowhere defined operator), as well as additional new operators H (halt) and *Ret* (return to check conditions).

Reconfigurable dynamically changing, many-sorted algebraic system is defined as follows:

$RAS = (\boldsymbol{A}, P, F, I_{F0}, I_{P0}, Rules, \boldsymbol{M}, Z, Rrules)$,

where *Rrules* – a set of "reconfigurable" rules that change the structure or the configuration of a given formal model. The rules from the given set *Rrules* are actually very similar to the ones from the original set *Rules* but relate to other properties of the model.

Algebraic systems identified in many studies. For example, algebraic systems reviewed in the monograph [5]. Many-sorted algebraic systems are described in the monograph [6]. A similar concept of evolving algebras, or, in modern interpretation, of abstract state machines (ASMs), was introduced by the American mathematician Y. Gurevich, as well as the concept of updates of predicates and functions [7]. It is well-known that ASMs can simulate "step-by-step" any type of machines (Turing machines, RAMs, etc.). Various uses of ASMs are also described in the works [8, 9]. A systematic study of ASMs is also done in [8]. One of the modifications of ASM has been described in [9]. The authors modified Gurevich's given notion of ASM to that of EMA which is the (Evolving Multi-Algebra) simply by replacing the originally given program (which is a syntactic object) by a semantic one: a function which has to be very simply definable over the static part of the ASM.

The main difference is that the reconfiguration of the structure of our model is carried out in the course of its execution. Another major difference between the interpretation of the ASM and interpretation of a tuple N as the network of abstract modules, or abstract machines (NAM) is structuring by using the algebra of algorithms by Glushkov V. M. [10, 11] which is also applicable to various problems of his colleagues and followers [12-17]. In particular, in this article we are using the superposition of "alpha-disjunction": $[\alpha](A \vee B)$ – which is an analogue of the operator "if", and the "composition: $A \bullet B$ or can be simply just A,B" operations for the structured description of the abstract machines networks. Also an additional operation "$\leftarrow$" is used for modificating the value of a variable, function or a predicate. "Reconfigurable" version of NAM we will call abbreviated as RNAM.

3.1. Formal NAM and RNAM description of the reconfigurable (deterministic or non-deterministic) Turing machines (RDTM and RNDTM)

The description of a Turing machine can be found, for example, in [18]. The work of the deterministic or non-deterministic Turing machine (NDTM) can be described by the following expression for the abstract module of NAM:

$M = Place(Head) \leftarrow 0;\ State(q_1) \leftarrow \mathbf{true};$
$[(\exists_{Any}\, q)\ State(q)]([a \leftarrow Tape(Place(Head))]$
$([\exists_{Any}(q, a, q', a', w')Program(q, a, q', a', w')]$
$(\{[State(q')\&S_{Q0}(q')](H \vee E), Tape(Place(Head)) \leftarrow a',$
$State(q) \leftarrow \mathbf{false}, State(q') \leftarrow \mathbf{true}, \{[w' = R]$
$(Place(Head) \leftarrow Inc(Place(Head)) \vee$
$[w' = L](Place(Head) \leftarrow Dec(Place(Head)) \vee E))\}, Ret\} \vee N) \vee N) \vee N),$

where q – the current state, $q \in Q$, $Q = \{q_1, q_2, \ldots, q_n\}$;

a – the symbol in the current cell of the tape, $a \in A$, $A = \{a_1, a_2, \ldots, a_m\}$;

q_1 – the initial state;

q' – the next state, $q' \in Q$;

a' – the next symbol in the current cell of the tape, $a' \in A$;

$w' \in (R, L, S)$ – the variable, that is determining the movement's direction of the head – right (R), left (L), stand on the observed cell (S);

$\exists_{Any}$ – the operator of non-deterministic selection of one of the tuples that are in the truth domain of the predicate *Program*;

Program(q, a, q', a', w') – the predicate, the truth domain of which defines the work program of a Turing machine, *Program*: $(Q\times A)\times(Q\times A\times W) \rightarrow \{$**true**, **false**$\}$;

Tape – the unary function, that defines a sequence of symbols on the tape,

Tape: $\mathbf{Z} \rightarrow A$, where $\mathbf{Z}$ – the set of integers;

Place – the unary function, that defines the head position,

Place: $Hd \rightarrow \mathbf{Z}$, where Hd – a set of head's names (in this case $Hd = \{Head\}$, where *Head* is the individual constant, or the name of the head);

State(q) – this predicate determines the state of the Turing machine (so that at any time the statement $(\exists!q)$*State*(q) is true, *State*: $Q \rightarrow \{$**true**, **false**$\}$);

Inc (*Dec*) – the increment (or decrement) function;

$S_{Q0}(q)$ – a characteristic function of a subset Q_0 of final states.

A certain symbol of the external alphabet $A = \{a_1, a_2, \ldots, a_m\}$ is called a dummy symbol. This symbol is represented by # or λ.

According to the [18] at any instant of time the controlling device is in a certain state q_i, which belongs to an internal alphabet or a set $Q = \{q_1, q_2, \ldots, q_n\}$. Disjoint subsets Q_1 and Q_0 of initial and final states respectively define the start and stop operations of a Turing machine. A Turing machine stops when $q \in Q_0$ after the execution of some command. We also introduce $S_{Q0}(q)$ – a characteristic function of a subset Q_0 of final states.

The list of all quintuples or commands (q, a, q', a', w') in the truth domain of the predicate *Program* which determines the operations of the Turing machine is a program of this machine.

In our example the quintuple (q_i, a_j, q', a', w') begins with the pair (q_i, a_j); $i = 1, 2, \ldots, n$; $j = 1, 2, \ldots, m$; n – the number of states; m – the number of symbols of the external alphabet A. For a program of the deterministic Turing machine (DTM) there should be only one such the quintuple. In the case of the non-deterministic Turing machine (NDTM) may be more than one tuple, starting with a pair of the form (q_i, a_j). The values of q_i and a_j are defined as follows:

$q_j = (\exists_{Any}\, q)$ *State*(q); $a_j =$ *Tape*(*Place*(*Head*)).

The values of q_i and a_j determined in this way immediately during the execution of the expression M.

Reconfigurable (deterministic or non-deterministic) Turing machine (RDTM and RNDTM) may further contain reconfigurable rules that change the structure or the configuration of a formal model. For example, it is possible to change the program of work of the Turing machine with the following rules from the set *Rrules*:

Program(q_i, a_j, q_1, a_1, L) ← **false**;

Program(q_i, a_i, q_2, a_2, R) ← **true**.

By incorporating these rules to the expression M we obtain the following expression RM for the abstract module of RNAM:

$RM =$ *Place*(*Head*) ← 0; *State*(q_1) ← **true**;
[$(\exists_{Any}\, q)$ *State*(q)]([a ← *Tape*(*Place*(*Head*))]
([$\exists_{Any}$(q, a, q', a', w')*Program*(q, a, q', a', w')]
({[*State*(q')&$SQ_0(q')$]($H \vee E$), *Tape*(*Place*(*Head*)) ← a',
State(q) ← **false**, *State*(q') ← **true**, {[$w' = R$]
(*Place*(*Head*) ← *Inc*(*Place*(*Head*))∨
[$w' = L$](*Place*(*Head*) ← *Dec*(*Place*(*Head*))∨ E))},
{*Program*(q_i, a_j, q_1, a_1, L) ← **false**;
Program(q_i, a_i, q_2, a_2, R) ← **true**}, *Ret*}∨ N) ∨ N)∨ N).

Replacing of the currently executed command may be performed as follows:

$RM' =$ *Place*(*Head*) ← 0; *State*(q_1) ← **true**;
[$(\exists_{Any}\, q)$ *State*(q)]([a ← *Tape*(*Place*(*Head*))]

$([\exists_{Any}(q, a, q', a', w')Program(q, a, q', a', w')]$
$(\{[State(q')\&SQ_0(q')](H \vee E), Tape(Place(Head)) \leftarrow a',$
$State(q) \leftarrow \mathbf{false}, State(q') \leftarrow \mathbf{true}, \{[w' = R]$
$(Place(Head) \leftarrow Inc(Place(Head)) \vee$
$[w' = L](Place(Head) \leftarrow Dec(Place(Head)) \vee E)\},$
$\{Program(q, a, q', a', w') \leftarrow \mathbf{false};$
$Program(q, a, q_2, a_2, R) \leftarrow \mathbf{true}\}, Ret\} \vee N) \vee N) \vee N).$

Now it is easy to describe the multitape non-deterministic Turing machine described in the books [21, 22]. The multitape non-deterministic Turing machine operates in accordance with the following expression for the abstract machine module:

$MK = Place(Head^{(1)}) \leftarrow 0; Place(Head^{(2)}) \leftarrow 0; \ldots; Place(Head^{(k)}) \leftarrow 0;$
$State(q_1) \leftarrow \mathbf{true}; [(\exists_{Any}\, q)\, State(q)]([a^{(1)} \leftarrow Tape^{(1)}(Place(Head^{(1)}))]$
$([a^{(2)} \leftarrow Tape^{(2)}(Place(Head^{(2)}))] \ldots$
$\ldots ([a^{(k)} \leftarrow Tape^{(k)}(Place(Head^{(k)}))]$
$([\exists_{Any}(q, a^{(1)}, a^{(2)}, \ldots, a^{(k)}, q', a'^{(1)}, a'^{(2)}, \ldots, a'^{(k)}, w'^{(1)}, w'^{(2)}, \ldots, w'^{(k)})$
$Program(q, a^{(1)}, a^{(2)}, \ldots, a^{(k)}, q', a'^{(1)}, a'^{(2)}, \ldots, a'^{(k)}, w'^{(1)}, w'^{(2)}, \ldots, w'^{(k)})]$
$(\{[State(q')\&S_{Q0}(q')](H \vee E),$
$Tape^{(1)}(Place(Head^{(1)})) \leftarrow a'^{(1)},$
$Tape^{(2)}(Place(Head^{(2)})) \leftarrow a'^{(2)}, \ldots, Tape^{(k)}(Place(Head^{(k)})) \leftarrow a'^{(k)},$
$State(q) \leftarrow \mathbf{false}, State(q') \leftarrow \mathbf{true},$
$\{[w'^{(1)} = R](Place(Head^{(1)}) \leftarrow Inc(Place(Head^{(1)})) \vee$
$[w'^{(1)} = L](Place(Head^{(1)}) \leftarrow Dec(Place(Head^{(1)})) \vee E))\},$
$\{[w'^{(2)} = R](Place(Head^{(2)}) \leftarrow Inc(Place(Head^{(2)})) \vee$
$[w'^{(2)} = L](Place(Head^{(2)}) \leftarrow Dec(Place(Head^{(2)})) \vee E))\}, \ldots$
$\ldots, \{[w'^{(k)} = R](Place(Head^{(k)}) \leftarrow Inc(Place(Head^{(k)})) \vee$
$[w'^{(k)} = L](Place(Head^{(k)}) \leftarrow Dec(Place(Head^{(k)})) \vee E))\}, Ret\} \vee N) \vee N) \ldots \vee N)$
$\vee N) \vee N),$

where superscripts correspond to the numbers of heads and tapes in multitape non-deterministic Turing machine. Description of such a reconfigurable (deterministic or non-deterministic) Turing machine is obvious.

3.2. Formal NAM and RNAM description of the reconfigurable (deterministic or non-deterministic) finite state automata (RFSA and RNDFSA)

The present study is based on the works [19-22]. The work of the deterministic or non-deterministic finite state automata (FSA or NDFSA) can be described by the following expression for the abstract module of NAM:

1) for Mealy NDFSA automata:

$A_1 = Place(XHead) \leftarrow 0; Place(YHead) \leftarrow 0; State(a_1) \leftarrow \mathbf{true};$
$[(\exists_{Any}\, a)\, State(a)]([x \leftarrow Xtape(Place(Xhead))]$
$([\exists_{Any}(a, x, a', y)Table(a, x, a', y)]$
$(\{[State(a')\&S_{AF}(a')](H \vee E), State(a) \leftarrow \mathbf{false}, State(a') \leftarrow \mathbf{true},$
$Ytape(Place(Yhead)) \leftarrow y,$
$Place(Xhead) \leftarrow Inc(Place(Xhead)),$
$Place(Yhead) \leftarrow Inc(Place(Yhead)), Ret\} \vee N) \vee N) \vee N),$

where a – the current state ($a \in A$, $A = \{a_1, a_2, \ldots, a_n\}$ – a finite set of internal states, a_1 – an initial state);

x – the symbol in the current cell of the input tape ($x \in X$, $X = \{x_1, x_2, \ldots, x_m\}$ – a finite set of input symbols;);

a' – the next state $a' \in A$;

y – the output symbol ($y \in Y$, $Y = \{y_1, y_2, \ldots, y_k\}$ – a finite set of output symbols);

$\exists_{Any}$ – the operator of non-deterministic selection of one of the tuples that are in the truth domain of the predicate *Table*;

Table(a, x, a', y) – the predicate, the truth domain of which defines the work table of NDFSA automata, *Table*: $A \times X \times A \times Y \rightarrow \{$**true**, **false**$\}$;

Xtape – the unary function, that defines a sequence of input symbols on the input tape, *Xtape*: $\mathbf{Z} \rightarrow X$, where $\mathbf{Z}$ – the set of integers;

Ytape – the unary function, that defines a sequence of output symbols on the output tape, *Ytape*: $\mathbf{Z} \rightarrow Y$;

Place – the unary function, that defines the heads positions, *Place*: $Hd \rightarrow \mathbf{Z}$, where Hd – a set of head's names (in this case $Hd = \{XHead, YHead\}$, where *XHead*, *YHead* are the individual constants, or the names of the input and output heads);

State(a) – this predicate determines the state of automata (so that at any time the statement $(\exists!a)State(a)$ is true, *State*: $A \rightarrow \{$**true**, **false**$\}$);

Inc – the increment function;

A_F – a subset of the final states, $A_F \subset A$;

$S_{AF}(a)$ – a characteristic function of a subset A_F of final states, S_{AF}: $A \rightarrow \{$**true**, **false**$\}$;

2) for Moore NDFSA automata:

A_2 = *Place*(*XHead*) ← 0; *Place*(*YHead*) ← 0; *State*(a_1) ← **true**;
[($\exists_{Any}$ a) *State*(a)]([x ← *Xtape*(*Place*(*Xhead*))]
([$\exists_{Any}$(a, x, a', y)*Table'*(a, x, a', y)]
({[*State*(a')&S_{AF}(a')]($H \vee E$), *State*(a) ← **false**, *State*(a') ← **true**,
Place(*Xhead*) ← *Inc*(*Place*(*Xhead*)),
Place(*Yhead*) ← *Inc*(*Place*(*Yhead*)),
Ytape(*Place*(*Yhead*)) ← y, *Ret*}$\vee N) \vee N) \vee N$);

the work of the deterministic or non-deterministic reconfigurable finite state automata (RFSA or RNDFSA) can be described by the following expression for the abstract module of RNAM:

3) for Mealy RNDFSA automata:

A_3 = *Place*(*XHead*) ← 0; *Place*(*YHead*) ← 0; *State*(a_1) ← **true**;
[($\exists_{Any}$ a) *State*(a)]([x ← *Xtape*(*Place*(*Xhead*))]
([$\exists_{Any}$(a, x, a', y)*Table*(a, x, a', y)]
({[*State*(a')&S_{AF}(a')]($H \vee E$), *State*(a) ← **false**, *State*(a') ← **true**,
Ytape(*Place*(*Yhead*)) ← y,
Place(*Xhead*) ← *Inc*(*Place*(*Xhead*)),
Place(*Yhead*) ← *Inc*(*Place*(*Yhead*)),
{*Table*(a, x, a', y) ← **false**,
Table(a, x, a_1, y_1) ← **true**}, *Ret*}$\vee N) \vee N) \vee N$),

4) for Moore RNDFSA automata:

A_4= *Place*(*XHead*) ← 0; *Place*(*YHead*) ← 0; *State*(a_1) ← **true**;
[($\exists_{Any}$ a) *State*(a)]([x ← *Xtape*(*Place*(*Xhead*))]
([$\exists_{Any}$(a, x, a', y)*Table'*(a, x, a', y)]
({[*State*(a')&S_{AF}(a')]($H \vee E$), *State*(a) ← **false**, *State*(a') ← **true**,
Place(*Xhead*) ← *Inc*(*Place*(*Xhead*)),
Place(*Yhead*) ← *Inc*(*Place*(*Yhead*)),
Ytape(*Place*(*Yhead*)) ← y
{*Table*(a, x, a', y) ← **false**,
Table(a, x, a_1, y_1) ← **true**}, *Ret*}$\vee N) \vee N) \vee N$).

3.3. Formal NAM and RNAM description of the reconfigurable information-inhibitor Petri net (RPNII)

A Petri net [23-26] is one of several mathematical modeling languages for the description of distributed systems. A Petri net is nothing but a directed bipartite graph, in which the nodes represent transitions (i.e. events that may occur and signified by bars) and places (i.e. conditions that are signified by circles). The directed arcs describe which places are pre- or postconditions for which transitions (signified by arrows).

Petri net usually is defined by a tuple:

$N = (P, T, F, H, M_0)$,

where $P = \{p_1, p_2, \ldots, p_n\}$ – a non-empty finite set of positions, $T = \{t_1, t_2, \ldots, t_m\}$ – a non-empty finite set of transitions, F, H – the incidence functions, M_0 – a function of the initial marking of the positions. We consider the first-order logical interpretation of 1-safe (or 1-bounded) Petri nets, for which three predicates are determined:

$F: P \times T \to \{\mathbf{true}, \mathbf{false}\}$; $H: T \times P \to \{\mathbf{true}, \mathbf{false}\}$; $M_0: P \to \{\mathbf{true}, \mathbf{false}\}$.

We shall also use well known in the literature on Petri nets two optional binary predicates

$F_{Inh}: P \times T \to \{\mathbf{true}, \mathbf{false}\}$ and $F_{Inf}: P \times T \to \{\mathbf{true}, \mathbf{false}\}$ in order to set the inhibitory and information arcs in a Petri net. Thus, information-inhibitor Petri net is defined as a dynamic algebraic system by a tuple: $NII = (P, T, F, F_{Inh}, F_{Inf}, H, M_0, R)$, where R is a finite set of rules, that can update the unary predicate $M(p)$ (current marking). In the terms of Petri nets theory such rules are called the rules of transitions firing. For this reason $|R| = |T|$.

The reconfigurable information-inhibitor Petri net now is defined as a reconfigurable dynamic algebraic system by the next tuple: $RNII = (P, T, F, F_{Inh}, F_{Inf}, H, M_0, \mathbf{R}, Rrules, S_{Upd})$, where *Rrules* is a set of the "reconfigurable" rules that can update the binary predicates F, F_{Inh}, F_{Inf}, H; S_{Upd} is a set of the rules that can update the sets P and T, if it is needed.

Further consideration is expedient to carry out with an example. Let's define a fragment of the network *NII* as follows: $F(p_1, t_1) = \mathbf{true}$, $F_{Inh}(p_2, t_1) = \mathbf{true}$, $F_{Inf}(p_3, t_1) = \mathbf{true}$, $H(t_1, p_4) = \mathbf{true}$, $M(p_1) = \mathbf{true}$, $M(p_2) = \mathbf{false}$, $M(p_3) = \mathbf{true}$, $M(p_4) = \mathbf{false}$. Rule r_1 of firing the transition t_1 is described by the following expression for the network NAM: $r_1 = [M(p_1)\&\neg M(p_2)\&M(p_3)](\{M(p_1) \leftarrow \mathbf{false},\ M(p_4) \leftarrow \mathbf{true}\} \vee E)$. After firing the transition t_1 marking becomes the following: $M(p_1) = \mathbf{false}$, $M(p_2) = \mathbf{false}$, $M(p_3) = \mathbf{true}$, $M(p_4) = \mathbf{true}$.

For reconfiguration of the fragment we introduce two positions p_5 and p_6 and define reconfiguration rules: $Rrules = \{F(p_1, t_1) \leftarrow \mathbf{false}, F_{Inh}(p_2, t_1) \leftarrow \mathbf{false}, F_{Inf}(p_3, t_1) \leftarrow \mathbf{false},$

$F_{Inh}(p_3, t_1) \leftarrow \mathbf{true}, F(p_5, t_1) \leftarrow \mathbf{true}, F_{Inh}(p_6, t_1) \leftarrow \mathbf{true}\}$.

The rule r' firing the transition t_1 is described now by the following expression for the network RNAM:

$r' = [M(p_1)\&\neg M(p_2)\&M(p_3)](\{M(p_1) \leftarrow \mathbf{false},\ M(p_4) \leftarrow \mathbf{true},$

$\{F(p_1, t_1) \leftarrow \mathbf{false}, F_{Inh}(p_2, t_1) \leftarrow \mathbf{false}, F_{Inf}(p_3, t_1) \leftarrow \mathbf{false},$

$F_{Inh}(p_3, t_1) \leftarrow \mathbf{true}, F(p_5, t_1) \leftarrow \mathbf{true}, F_{Inh}(p_6, t_1) \leftarrow \mathbf{true}\}\} \vee E)$.

The new rule r'' firing the transition t_1 is described now by the following expression for the modified network RNAM: $r'' = [\neg M(p_3)\& M(p_5)\& \neg M(p_6)](\{M(p_5) \leftarrow \mathbf{false},\ M(p_4) \leftarrow \mathbf{true}\} \vee E)$.

Another principle of Petri nets modification was proposed in [27] and was associated only with multiple arcs.

4. Conclusion

In this paper we proposed some "reconfigurable directly executable formal models" that are suitable for the specification of computing and networking in the cloud environment and other networks which are similar to wireless networks MANET. These models can be easily programmed and implemented in computer networks. In this article, it was demonstrated that the directly executable formal models can be classified as the models of intermediate type, possessing properties of logic-algebraic and executable models.

The major difference between the interpretation of the ASM and the network of abstract modules, or abstract machines (NAM) is the structuring by using the algebra of algorithms. Reconfigurable version of NAM we have called abbreviated as RNAM. Reconfigurable, dynamically changing, many-sorted algebraic system is the basis for the following models of discrete systems: the formal "directly executable" NAM and RNAM; the formal "directly executable" NAM and RNAM models for the reconfigurable (the deterministic and the non-deterministic) Turing machines (RDTM and RNDTM); the formal "directly executable" NAM and RNAM models of the reconfigurable (the deterministic and the non-deterministic) finite state automata (RFSA and RNDFSA); the formal "directly executable" NAM and RNAM models of the reconfigurable information-inhibitor in Petri nets (RPNII).

The modifications of some well-known formalisms will allow us in the future to construct adequate models of systems with variable structures. Preliminary considerations of this issue as well as the concept of the reconfigurable formal models were carried out in the article [28]. We propose to focus further attention on the application of the proposed models in the field of cloud computing, which is characterized by the following problem.

Moving traditional applications and their infrastructure to the cloud has shifted the in-house control issue to a third party auditor. It posts many challenges including the security and the privacy issues which come on top along with the performance and the availability out of the security which is the number one concern. Clearly using the concept of cloud computing does not make the security issues go away. It becomes an even challenging topic. So In that case, it is not a quite usual utility concept that we are talking about in here. Still there is a problem concerning the security which is the data leakages due to the poor authentications and information's assurance. In this research study our investigation presents a framework and an efficient construction for a significant integration of these two components in our protocol design along with the present of the new approach to keep an efficient and secure cloud computing framework. Juels et al. in [29] they have described the concept of the "proof of retrievability" (POR) model, where the concepts of spot-checking and error-checking and correcting codes are used to ensure both of the possession and the retrievability of data files on the archive service systems. Specifically, some special kind of blocks called the "sentinels" which they are randomly embedded into the data file F for the detection purposes, and F is further encrypted in order to protect the positions of these special blocks mentioned.

However, like [30], mentioned that there are some kind of defects in the system like the number of queries/ tasks a client can perform is also a fixed priori along with the introduction of pre-computed "sentinels" which prevent the development of the dynamic data updates. In addition, the concept of the public auditability is not supported in their proposed scheme. Shacham et al. in [31] designed an improved the POR scheme with full proofs of security and deployed in the security model defined in [29]. They are using some kind of publicly verified homomorphic authenticators [32], on which the proofs of security can be aggregated into a small authenticator value, and the required public retrievability is achieved. Still, the authors are only considering the static type of data files and not the dynamic type and their dynamic operations and this issue is still of course affecting the security of the scheme which is the main concern now of course along with the data availability and integrity.

Our proposals for further research (part of the problem was formulated by one of the authors of this paper in [4]) can be summarized as follows:

(1) We will propose a general formal proof of retrievability (POR) model with public verifiability for the cloud's data storage system, in which block less verifications are achieved and encountered.

(2) We will also equip our proposed proof of retrievability (POR) construction with the functions of supporting for the fully kinds of the dynamic data operations especially to support the block insertion mechanism, which is missing in the most currently existing schemes. In addition to this, multiple TPA's (Third Party Auditors) in working mode in order to handle all the requests in between the clients and the CSS (Cloud's Storage System).

(3) We will propose the new approach of checklist generation for each kind of the cloud like public cloud, community cloud and private cloud with an appropriate procedure. This approach is more affordable and efficient rather than using each time specific algorithms for providing the specific required security.

The objectives of our future scope are:

1) To motivate the public auditing system of data storage security in Cloud Computing, and to propose a protocol supporting for the fully kinds of the dynamic data operations, especially to support the block insertion mechanism, which is missing in most of the currently existing schemes.

2) To extend the scheme to support the scalable and the efficient public auditing in Cloud Computing using the IT auditing and the checklist generation for each kind of the cloud. In particular, the scheme achieves the concept of the batch auditing where multiple delegated auditing tasks from different users will be performed simultaneously and significantly fast by the TPAs.

3) To prove the security of our proposed construction and to justify the performance of the proposed scheme through some significant and concrete kinds of implementations and comparisons with the state-of-the-art.

Acknowledgments
The work is performed as part of the Federal Special Purpose Program - UIN: RFMEFI57414X0045).

References

[1] Mustafa Sadeq Jaafar, Sawant H. K. ACK Based Scheme for Performance Improvement of Ad-hoc Network // International Journal of Advances in Engineering & Technology, (IJAET). Vol. 3, Issue 2, May 2012.

[2] Mustafa Sadeq Jaafar, Sawant H. K. Design and Development of ACK-Based Scheme Using FSA for Ad-hoc Networks // International Journal of Modern Engineering Research, (IJMER). Vol. 2, Issue. 2, Mar-Apr 2012, pp. 102-106.

[3] Mustafa Sadeq Jaafar. On Modification ACK-Based Scheme using FSA for Ad-Hoc Networks // Proceedings of the Eleventh International Conference of Science and Technology "New Information Technologies and Systems", Penza, Penza State University, Russia, November 25–27, 2014, pp. 334-340.

[4] Mustafa Sadeq Jaafar. IT Auditing Based on Public Verifiability and the Cloud's Dynamics and Data Storage Security using Multiple TPA's // Proceedings of the XIX-th International Conference "University Education", Penza, Penza State University, Russia, April 9–10, 2015, pp. 198-202.

[5] Malcev, A. I. Algebraic Systems. Springer-Verlag, 1973, 329 p.

[6] Plotkin, B. I. Universal algebra, algebraic logic, and databases. Dordrecht – Boston: Kluwer Academic Publishers, 1994. 453 p.

[7] Gurevich, Y. Evolving Algebras 1993: Lipari Guide // Specification and Validation Methods. E. Boerger (ed.), Oxford University Press, 1995, pp. 9-36.

[8] Boerger, E. Unifying View of Models of Computation and System Design Frameworks // Annals of Pure and Applied Logic. Vol. 133, 2005, pp. 149-171.

[9] Grigorieff, S., Valarcher, P. Evolving Multialgebras Unify all Usual Sequential Models // Symposium on Theoretical Aspects of Computer Science. Nancy, France, 2010, pp. 417-428.

[10] Glushkov, V. M. Theory of Automata and Formal Transformations of Microprograms // Kibernetika, No. 5, 1975, pp.1-10.

[11] Gluschkow, W. M., Zeitlin, G. E., Justchenko, J. L. Algebra. Sprachen. Programmierung. Akademie-Verlag, Berlin, 1980. – 340 p.

[12] Tseytlin, G. E. Glushkov Algebras and Clone Theory // Cybernetics and Systems Analysis. Vol. 39, No. 4. Springer-Verlag, New York, 2003, pp. 509-516.

[13] Cejtlin, G., Jushchenko, E. Mathematical theory of multiprocessor control systems and its applications // Computer Science Journal. Vol. 2, No. 3(6), 1994, pp. 247-261.

[14] Cejtlin, G.E. Schematics of Structural Parallel Programming and its Applications // Mathematical Institute, Czechoclovak Academy of Olomous MFCS'79. Lecture Notes in Computer Science. Vol. 74, 1979, pp. 474-481.
[15] Ivanov, P. M. Algebraic modelling of complex systems. – Moscow, 1996. – 274 p.
[16] Nepejvoda, N. N. Abstract algebras of different classes of programs // Proceedings of the 3rd international conference on applicative computation systems (ACS'2012), 2012, pp. 103-128.
[17] Nepejvoda, N. N. Algebraic approach to control. Control Sciences, 2013, No. 6, pp. 2-14.
[18] Gavrilov, G. P., Sapozenko, A. A. Selected Problems in Discrete Mathematics. – M.: Mir Publishers, 1989. – 414 p.
[19] Brauer, W. Automaten Theorie. B. G. Teubner Stuttgart, 1984. – 392 p.
[20] Cooke, D. J., Bez H. E. Computer Mathematics. Cambridge University Press, Cambridge, 1984. – 384 p.
[21] Aho, A. V., Hopcroft, J. E. Ullman, J. D. The Design and Analysis of Computer Algorithms. Addison-Wesley Publishing Company, 1976. – 470 p.
[22] Aho, A. V., Ullman, J. D. The Theory of Parsing, Translation and Compiling. Vol. 1. Prentice-Hall, Englewood Cliffs, 1973. – 309 p.
[23] Petri, C. A. Introduction of General Net Theory // Lecture Notes in Computer Science. Berlin: Springer-Verlag. Vol. 84, 1980, pp. 1-26.
[24] Peterson, J. L. Petri Net Theory and the Modeling of Systems. Prentice-Hall, Englewood Cliffs, NJ, 1981. – 264 p.
[25] Petri Nets. Fundamental Models, Verification and Applications / Ed. by M. Diaz. John Wiley and Sons, 2009. – 613 p.
[26] Petri Nets. Theory and Applications / Ed. by V. Kordic. I-TECH Education and Publishing, Vienna, Austria, 2008. – 544 p.
[27] Valk, R. Self-modifying Nets, a Natural Extension of Petri Nets // Lecture Notes in Computer Science. Berlin: Springer-Verlag. Vol. 62, 1978, pp. 464-476.
[28] Mustafa Sadeq Jafar, Zinkin, S. A. Reconfigurable formal models for MANET and Cloud networking and computing // Proceedings of the XII-th International Conference of Science and Technology "New Information Technologies and Systems". Penza State University, Russia, 2015, pp. 141-149.
[29] Juels, A., Kaliski, B. S., Jr., "Pors: proofs of retrievability for large files," in Proc. of CCS'07. New York, NY, USA: ACM, 2007, pp. 584–597.
[30] Bowers, K. D., Juels, A., Oprea, A. Proofs of retrievability: Theory and implementation // Cryptology ePrint Archive, Report 2008/175, 2008.
[31] Shacham, H., Waters, B. Compact proofs of retrievability // In Proc. of ASIACRYPT'08. Springer-Verlag, 2008, pp. 90–107.
[32] Boneh, D., Lynn, B., Shacham, H. Short signatures from the weil pairing // In Proc. of ASIACRYPT'01. London, UK: Springer-Verlag, 2001, pp. 514–532.