G. Balázsfalvi and J. Sztrik[1]

A Tool for Modeling Distributed Protocols

Gábor Balázsfalvi was born in 1981 in Hungary. He graduated in 2005 and gained an Msc. degree as programmer mathematician from University of Debrecen. He is PhD student both at University of Debrecen, Hungary and at University Rovira i Virgili, Spain. Recently his research interests include modeling techniques, distributed algorithms, peer to peer networks, cellular automata.

János Sytrik is a Full Professor at the Faculty of Informatics, University of Debrecen, Hungary. He obtained the Candidate of Mathematical Sciences Degree in Probability Theory and Mathematical Statistics in 1989 from the Kiev State University, Kiev, USSR, Habilitation from University of Debrecen in 1999, Doctor of the Hungarian Academy of Sciences, Budapest, 2002. His research interests are in the field of production systems modeling and analysis, queueing theory, reliability theory, and computer science.

ABSTRACT

Recent peer to peer protocols have mainly been investigated and tested by discrete-event network simulators, e.g. ns-2. These software products usually model the network topology in great details, thus giving reliable picture about a possible run of the modeled protocol. In most cases they can estimate performance measures like throughput, traffic intensity, probability of failure, etc. These protocols are often analyzed in many ways, non of them are standard. In this paper, we elaborate on a new approach based on cellular automata designed to prove correctness of distributed protocols in some cases. Cellular automata are well known for their complex global behaviors, although they are constructed from simple and similar pieces. We shall demonstrate how standard distributed protocols can be modeled by cellular automata. The correctness of the protocols can also be proved by this technique, or – in some cases – counter-examples may be found. Our approach is supported by a simulator that can help researchers to model their protocols by the proposed method.

Keywords:
protocols, computer network performance, discrete event simulation, peer to peer, modeling, cellular automaton

1 Research is partially supported by Hungarian Scientific Research Fund-OTKA K 60698/2006, OTKA T049409, the MTA-DFG Fund 2006, and grants of Xerox Foundation UAC (No. 1478-2004),U.S.A..

1 INTRODUCTION

Modeling protocols and algorithms used by distributed systems is important in order to prove their trustworthiness. This is unavoidable for some sorts of applications, for example, where high level of security or reliability are needed. These protocols are widely used in today's information technology, for example in peer-to-peer networks, which are very popular due to their benefits as compared to the regular client-server conception.

Recently more and more discrete event simulators have been developed aiming at evaluating performance measures of peer to peer systems [2], [3], [4]. Classic network simulators like ns-2 [1] can be used for this task as well. These tools can be classified according to the level they can describe a network, the number of network nodes they can handle, and the performance measures they can estimate. In the very usual case when we know for sure that protocol in question works well, these tools are very useful to know the characteristics of the designed system. However, there might be cases when something goes wrong, and the discrete event simulation fails to show what was expected. We can enumerate several reasons for that: probably we have selected a wrong type of simulator, we have made some mistakes in the programming of the simulator, or we simply have a wrong algorithm for our problem.

In this paper we propose a new approach by the help of which we can be able to prove or refute mathematically the correctness of a given protocol by mapping the investigated network to a cellular automaton [22]. Each cell behaves like a node in the network, a cell that is only allowed to communicate directly with its neighbors. The protocol under consideration can be programmed into the transition function of the cells. By doing so, we can make benefit of the theoretical results known about cellular automata because we can contrast or parallel them with the goals of the modeled protocol.

A cellular automata simulator tool has been developed to help understanding and visualizing the described approach. Some known protocols have been modeled by this cellular automata methotology, too. The paper ends with a conclusion and an outline of further research.

2 MEASURES OF DISTRIBUTED PROTOCOLS

In a real network environment, the measures of algorithms that can be evaluated are usually the following: running time (real time in seconds), network throughput, traffic overhead, bandwidth utilization, bottleneck analysis. For these performance measures, wherever it is possible, we first try to use formal methods, like queueing theory [12]. Simple queues are usually distinguished by the type of the arrival process (e.g., Markovian inter-arrivals or general and independent arrivals); the service time distribution (e.g. arbitrary distributed service time); the number of servers in the queue; the capacity of the queue; system population, i.e. the maximum number of jobs that can arrive at the queue, also referred to as source-size; queueing discipline (e.g. First In First Out). From simple queues we can build more complicated networks for that we know formal methods to evaluate their performance measures. We often make simulations, when formal methods fail to analyze such systems due to their complexity. We can make more exact measurements with network emulators based on real traces [11], or we can try out the algorithm under investigation in a closed test network environment [5]. Of course, formal evaluation is preferred against simulation, and simulation is preferred against emulation or try-out because on one hand they are much faster, and on the other hand they give more exact results

In this paper however, we aim at discussing much different measures. First of all, we are going to investigate algorithms on a more abstract level being independent of every implementation issues. The measures that are usually evaluated for abstract algorithms are the following:

- Correctness: We want to know if the procedure under consideration does what it is intended to. For this reason, we set up the goals of the algorithm, and prove that it is reached in finite time.
- Time complexity: We have proved that the algorithm is correct, and now we would like to know the time needed to achieve the goals set up previously. This time is not a real time measure, but an abstract time e.g. the steps of the algorithm.
- Communication complexity: By this quantity we measure the amount of information passed from one node to another; or the amount of information passed in total in the system during the algorithm. This can be defined e.g. by counting the total number of state changes, too.
- Space complexity: It might denote the space where information is temporarily stored, or the states of the nodes during the algorithm.

These measures are often evaluated by analyzing the protocol itself and describing it with pseudo-code in some formal programming language. For sequential algorithms we very frequently choose a Turing machine implementation to dissect. Distributed algorithms cannot be well examined by Turing machines, because the participant nodes work in parallel and more or less independently. For this reason, we propose a parallel model by which we can study the behavior of such algorithms. This model is called cellular automaton and will be described in the next section where we will show examples of applying this model to distributed protocols like broadcasting, state-synchronization and others, too.

3 CELLULAR AUTOMATA

3.1 Definitions

Cellular automata can be very different distinguished by lattice type and dimension, type of neighborhood, type of transition function, and type of state set. Most of the next definitions follow the paper of Toffoli [21].

Definition 1 (Displacement) *Let $S = Z^n$ be the commutative group of translations of the n-dimensional lattice I. An $s \in S$ is called displacement. A displacement can be applied on a place $i \in I$ resulting a new place $i + s$. The – operator on two places i_1 and i_2 results a translation $s = i_1 - i_2$ such that $i_1 = i_2 + s$. The elements of the lattice will also be referred to as cells.*

Definition 2 (Neighborhood) *A neighborhood is a finite set of displacements.*

We can define the + application operator on a neighborhood and a place, i.e. if $i \in I$ is a place and $N \subset S$ is a neighborhood, then $i + N = \{i + s \mid s \in N\}$. It is clear that $i + N$ is a finite set of places. The elements of $i + N$ are called neighbors of i.

Definition 3 (Configuration) *Let A denote a finite alphabet; we will refer to it as state set. Moreover let I be a lattice. The set of configurations is the product of "copies" of A indexed by the elements of I, i.e. $Q = A^I$. It is important that, although we denote the state set by an alphabet, the set itself can be of any complexity, e.g. a cross product of other sets, etc. The main point is that the cardinality of the state set is finite.*

Definition 4 (State) *If q is a configuration, i is a place, then $q[i]$ is called the state of i.*

Definition 5 (Transition) *A local transition $\lambda = (N, f)$ is a pair of a neighborhood (N) and a mapping (f) $f : A^N \to A$. It determines the next state of a place i by applying f on places $i + N$. A local transition applied on all the places of a configuration q yields a new configuration q'. This relation between q and q' is called global transition function induced by the local transition function, and it is denoted by τ.*

Definition 6 (Cellular automaton) *A cellular automaton is a lattice, a neighborhood, the state set and the local transition function together.*

Sometimes we distinguish a state called quiescent state. This special state q_0 is stable, i.e. $f(q_0,\dots,q_0) = q_0$. Once a cell entered, it can never leave the quiescent state. A configuration is finite if only a finite number of cells are non-quiescent. In applications for distributed protocols, we are rather interested in finite configurations.

3.2 Neighborhoods on Euclidean coordinates

The n-tuple $(z_0, z_1, \dots z_{n-1})$, where $z_i = \in Z$, forms a coordinate in the n dimensional Euclidean space Z^n. In the two dimensional space, we differentiate triangular, square and hexagonal lattices. We use the square lattice the most often. Although we can define any finite set of displacements as a neighborhood, we very often use the following ones, which can be defined without modification on other lattices and in any dimensions as well.

Definition 7 (Moore neighborhood)

$$M_k^n = \left\{ m \in Z^n \mid k \geq \max_{i=0}^{n-1} \mid m_i \mid \right\}.$$

Definition 8 (Von Neumann neighborhood)

$$H_k^n = \left\{ m \in Z^n \mid k \geq \sum_{i=0}^{n-1} \mid m_i \mid \right\}.$$

These neighborhoods include the actual cell itself. The Moore neighborhood includes the cells the coordinates of which differ from the actual cell's coordinate in any member, but the difference is at most k. In one dimension and for $k = 1$, these cells are the left and the right neighbors, while in two dimensions, if the actual cell has the coordinate (0,0), its neighborhood contains (–1,–1), (0,–1), (1,–1), (–1,0), (0,0), (1,0), (–1,1), (0,1), (1,1). There is a modified version where only the displacements with all nonnegative members play role. The other important neighborhood is named after Von Neumann. This includes every cell the coordinate of that does not differ from the actual cell's coordinate in more than k in sum. In one dimension and for $k = 1$ this means the same as before, but in two dimensions, it contains only 5 cells: (0,–1), (–1,0), (0,0), (1,0), (0,1). We show these examples in Fig. 1.

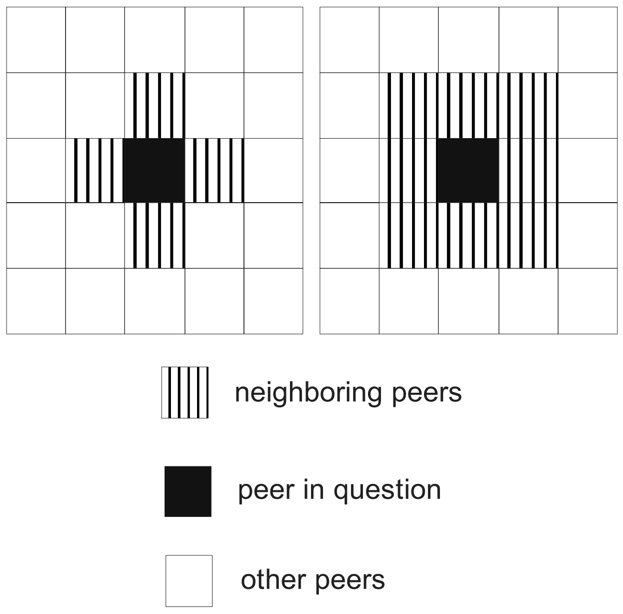


Fig. 1 Von Neumann and Moore Neighborhoods

In one dimension we very often use another important topology, namely the ring one. In this case, we have finitely many (e.g. n) cells in one row, whose places are denoted by $0,1,\ldots,n-1$. The cell on the i^{th} place has a neighborhood of size three, consisting of the cells on the places $(i-1)\bmod n$, i, and $(i+1)\bmod n$.

4 MODELING TOOL

To help the modeling and evaluation processes, we can use a cellular automata simulator tool. Although there are free tools available for download and modification we have decided to develop our own tool programmed in Java for this task. It is very flexible in terms of coordinate systems, lattices, neighborhoods, state and rule sets. Of course, only finitely many cells can be modeled, but this is sufficient for most of the applications. In this section we show what components are necessary to implement in order to test an algorithm.

The algorithm under investigation must be represented by a Java class implementing the *camodeler.Program* interface. It has only one method called *nextState*, which takes a *Cell* object and creates a new *State* object (it should not modify the actual state itself). The objects of the *Cell* class represent the cells and they have methods for getting and setting the actual state or neighborhood.

We can create special neighborhoods by implementing the *Neighborhood* interface and its *getNeighboringCells* method, that should fill in a *Collection* of *Coordinates* with the coordinates of the neighbors of the cell that is place at the coordinate given as parameter. By default we implemented the 2-dimensional euclidean coordinate space, and the one dimensional ring topology. If we need different space, we must implement the *CoordinateFactory* interface that can create the coordinates of that space. At the first invocation, the *createCoordinates* method is called with a *null* parameter, after that it is given the previously created coordinate. This factory object must give a *LayoutManager* too, which is responsible to place the cells of the created coordinates, and the specific coordinate should provide with the actual layout number by the *getPreferredLayoutNumber* method.

The *State* itself is only an interface, too. Its implementation should represent the members of the state set of the cells. It is necessary to give a method for copying the state itself; and another method that gives a so-called visualizer object. This is done by default by the *camodeler.program.DefaultState* class, which implements the cloning by the Java's built-in *Cloneable* interface, and the visualizer object is a simple *StateVisualizer* object, which shows the content of the state by putting the state's string representation to a *JLabel* object. If we need different visualization, we must extend the *StateVisualizer* class. We should re-implement the *setState* method; we can use the *JLabel* object (it is called *label*) of the *StateVisualizer* class, or choose a completely new way of visualization. In the latter case, we need to give back the new *JComponent* object with the *getVisualObject* method.

Finally, we need a main class that creates a *Program*, a *CoordinateFactory*, a *Neighborhood*, a *Cell*, a *CellularAutomaton* object and sets the initial states of the cells. By default, all the cells are identical and created as copies of the created cell (it needs a *State* object at creation). We can call the automaton's *setRewindStatus* to set up the initial conditions at which we can restart the simulation in the GUI of the software. Than we need to create a *Scheduler* object, and a *ModelerFrame* object of which *setVisible* method should be called for the GUI to show up.

5 EXAMPLES

In the following algorithms we need to introduce several restrictions on the network nodes and the communication. These are the following:

– The communication is free of failures.
– The communication is synchronized: every peer has a clock, each ticking at the same time, and the communication and state-change is done for each node between two consecutive ticks.
– Every node has the same neighborhood: this is a very strong restriction, but reasonable for algorithm-analysis. We can work however with any lattice in any dimension that makes the modeling comfortable.

5.1 Message Broadcasting

The node that has the message currently is denoted by cross and the others are shown as circles. The done state is denoted by a star. In this example we have a 2- dimensional grid, i. e. a rectangle of cells interconnected according to the 2-dimensional Von Neumann neighborhood. The cells represent peers.

A peer can directly communicate with the other peers that are symbolized by neighboring cells. One peer aims at delivering a message to all the other peers. For this reason, we use a special state in the state set that denotes the message. The transition function changes into that message state from its initial state if any of its neighbors possesses the message state, too. After having this state, in the next step, it processes the message, and changes its state to 'ready'. This can be seen in Fig. 2. By this very simple algorithm, the message travels through all the peers and reaches them only once. By using more states, the 'speed' of the message can be decreased to any rational number between 0 and 1. E.g. when it is 12, it moves one in every two steps. We can evaluate easily the measures pointed out in section 2. We have seen that the protocol is correct. The time complexity depends on the position of the initiator, but it is not more that $n + m$ for $n \times$ m nodes (it is in fact the diameter of the spanning tree generated by the broadcasting protocol [19], supposing that the initiator is the root of the spanning tree). The state set is as follows: we need to denote the idle state, the message, and the done states. We do not actually need more states, so 3 states per node are sufficient. The communication complexity is $2 \times n \times$ m, because every node changes its state exactly twice.

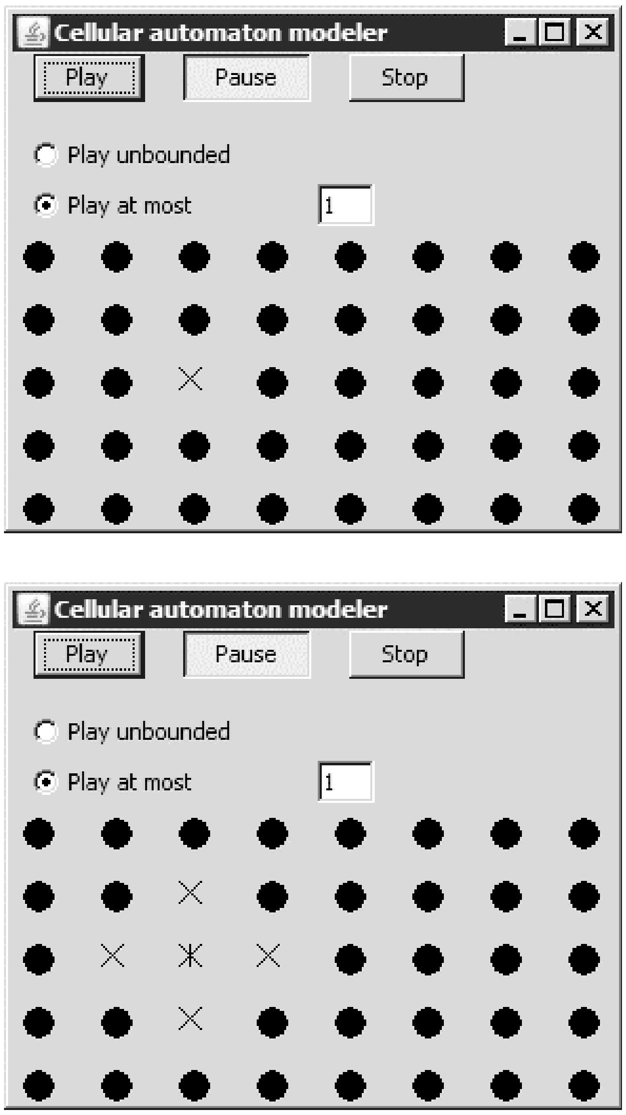


Fig. 2 Message broadcasting. The node that has the message currently is denoted by cross and the others are shown as circles. The done state is denoted by a star.

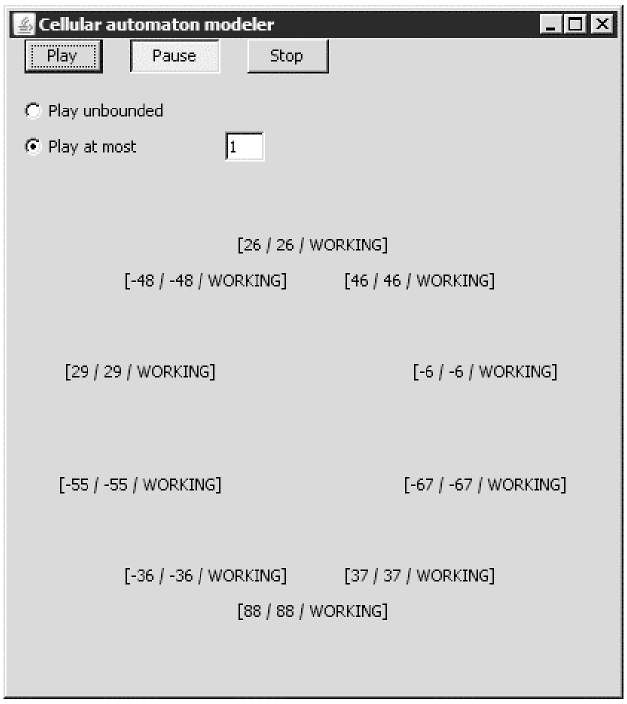
In this example the visualizer of the peers draws either a circle, a cross or a star depending on the current state. When the peer has the message then it is represented by a cross, when it receives the message, it becomes depicted as a star, respectively.

5.2 Spanning Tree

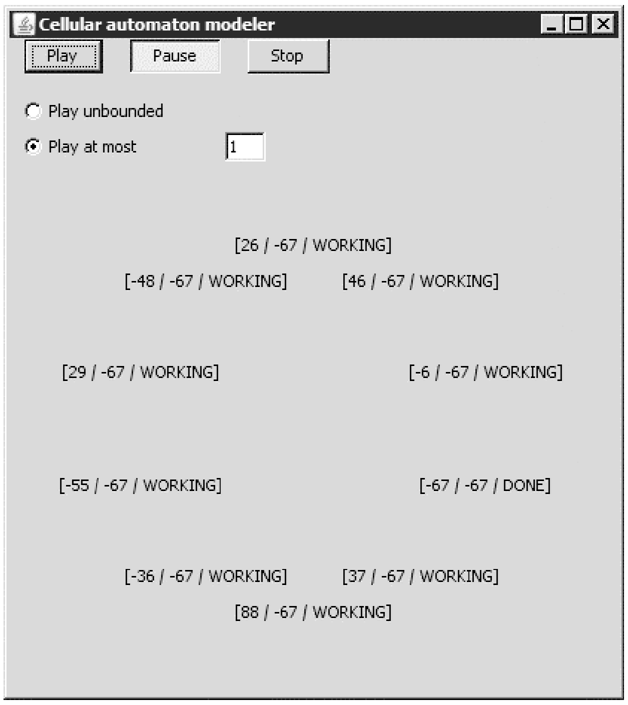
As it is known from [19] every algorithm for broadcasting is capable to build up a spanning tree of a network. There is another interesting result: No algorithm that would work independently of the number of initiators can build a spanning tree. This can be very easily proved by cellular automata. To do so, let us suppose that there is such a protocol, thus there is a cellular automaton, which can make a spanning tree. Let this automaton be based on a ring-lattice and suppose it consists of three nodes at least. Let us suppose that every peer initiates the protocol at the same time. This means that they all have the same state at the beginning. Since every node has two neighbors in a ring (plus itself as third), they all "see" the same states of the neighbors and they all enter the same state again. Thus every node will be in the same state forever. However, a spanning tree should be denoted in a different way at the root node than at an internal node, because the root node does not have a parent. This means that at least two nodes should be in different state at the end of the protocol, thus we got a contradiction.

5.3 Leader Election in Ring Networks

In the previous paragraph we have seen that having a unique leader among the peers of a network is often desirable. Now we take a look at an algorithm that peers can use to elect one of them in a ring topology. From the above-mentioned reasoning, it turns out that the peers should not be all in the same state. In the following algorithm we will suppose that the state set of each peer will consist of two integer numbers and a state flag. The state flag is set to 'working' for all of them at the beginning. One of the integers will denote a unique identifier picked at ran-



```
[26 / 26 / WORKING]
[-48 / -48 / WORKING]    [46 / 46 / WORKING]
[29 / 29 / WORKING]    [-6 / -6 / WORKING]
[-55 / -55 / WORKING]    [-67 / -67 / WORKING]
[-36 / -36 / WORKING]    [37 / 37 / WORKING]
[88 / 88 / WORKING]
```



[26 / -67 / WORKING]
[-48 / -67 / WORKING] [46 / -67 / WORKING]
[29 / -67 / WORKING] [-6 / -67 / WORKING]
[-55 / -67 / WORKING] [-67 / -67 / DONE]
[-36 / -67 / WORKING] [37 / -67 / WORKING]
[88 / -67 / WORKING]

Fig. 3 Election process.

dom, the other one is the minimum that the peer has seen so far. The idea is very simple: the peers elect the one with the minimal identifier. For this, each node looks at the preceding neighbor (in the clockwise order). If that node has seen a smaller value than the node in question, that minimal value is copied. Otherwise if the preceeding neighbor's minimal value is equal to the peer's identifier, the peer can be sure that it owns the minimal identifier because it has traveled around the ring. At this moment, it changes its state flag to 'done'. Given n nodes, the state complexity of this algorithm is $2 \times n^2$. The time complexity is n if we are ready when the leader knows about itself that it is elected. If not, we need another n steps to notify the others about the result. The communication complexity in the worst case – when the identifiers are in order – is n^2 while in the best case it is n only. The first step and the final one can be seen in Fig. 3.

5.4 Synchronization

The next example is a synchronization problem, often referred to as the "Firing squad" synchronization problem first solved by Moore. The solution can also be found in [22]. The steps of the algorithm can be seen in Fig. 4. From left to right we can see the time passing. We have some peers connected to each other according to the one dimensional Von Neumann neighborhood from bottom to top. Their job is to get to a common special state exactly at the same time without knowing how many they are. The time-line is shown in the figure from left to right, and we can see that in the end, the cells are in the same common state denoted by pure black color. We are going to solve this problem by the help of a super-peer. Therefore, the first cell, which is the bottommost one (it can be elected e.g. by the algorithm discussed above), sends a message with a speed of one towards the topmost peer. Moreover, it sends another message with a speed of 13 in the same direction. Afterwards, the first peer changes its state to the 'ready to fire' state, which is the last state before the common state they want to reach. When the first message reaches the topmost cell, that cell also enters the 'ready to fire' state, and sends the message back at the same speed. Eventually, this message meets the other one traveling slower at the cell in the middle, or at two neighboring central cells (as shown in the figure). The cell, where this happens is going to behave like the first one: it sends two messages, but this time in both directions (four messages in total). The cell changes its state to the 'ready to fire' state, too. This means that this peer becomes a super-peer as well as the bottommost. Sooner or later, every peer will be ready to fire due to this recursive method. What is more, there will not be any peer of that all the neighbors and itself as well are in the 'ready to fire' state, before all the peers are ready. When a cell and all of its neighbors are ready, that cell enters the common state. This happens to all peers at the same time, as the process is shown in Fig. 4. This figure illustrates the messages traveling through the cells. On the left hand side we can also see the first two messages, one traveling at speed 1 and the other one, which is slower and travels at speed 13. Clearly the time complexity of this algorithm is $3n$, if we have n nodes. However, the problem's time complexity is $2n - 2$.

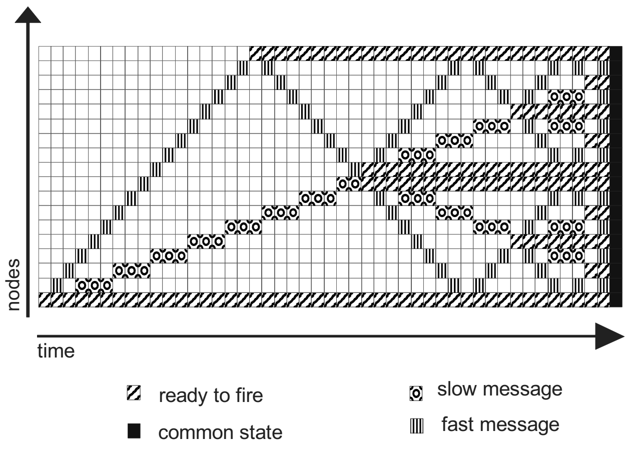


Fig. 4 Synchronization process.

5.5 BitTorrent Simulation

Our last example is the model of BitTorrent file sharing system. In BitTorrent the data is cut into hundreds of pieces. Clients download from one another these small junks of data in parallel. This time we have modeled only 50 pieces of data to be distributed among the participants. There are 16 peers and they are interconnected according to the 2-dimensional Von Neumann neighborhood. At start, only one has all the pieces contrarily to the others that do not have anything. As time passes , the neighbors copy parts of the pieces one after the other. This time the program of the cells is not deterministic. The participants select the current source and the current piece randomly. They also select at random a number from the set $\{0.1, 0.2, \ldots, 1\}$, and "download" that many ratio of the selected piece. The cells store real numbers for every piece, and increment the number of the selected piece by the selected number. Of course, they can only select pieces that are already present at the selected neighbor. This leads to a very high state complexity, however the algorithm can be programmed in a very easy way.

At last, the peers that do not even flank on the original owner will have all the pieces. A snapshot of this process can be seen in Fig. 5.

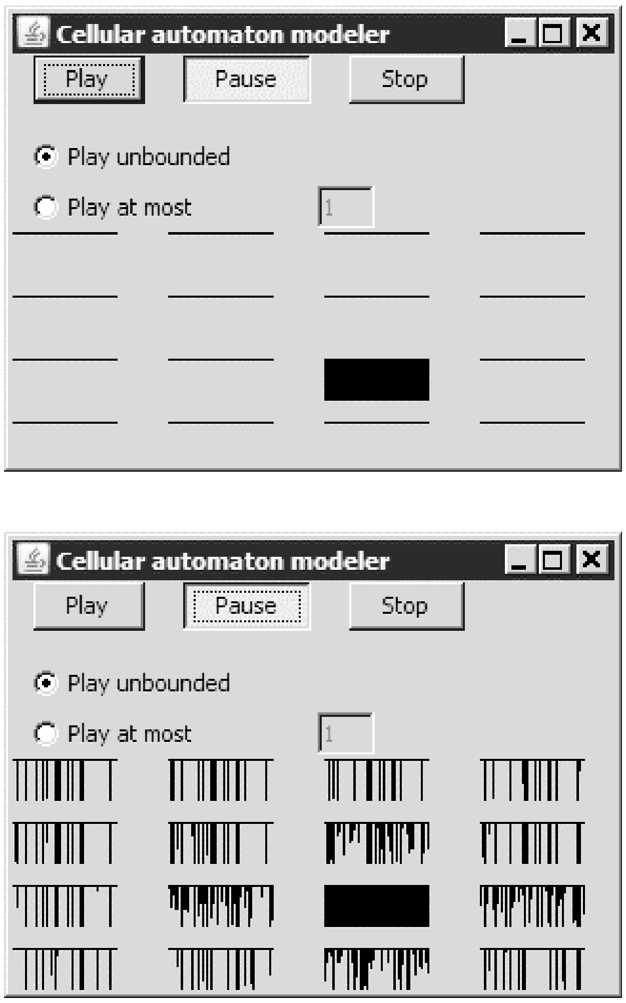


Fig. 5 BitTorrent simulation (start and halftime).

6 CONCLUSION AND FURTHER WORK

In this paper we have proposed a new modeling method for distributed protocols. By the help of this approach, we can evaluate time, state and communication complexity of certain distributed algorithms. Unfortunately, the model has some limitations: the nodes in the network are not free to connect to other nodes because the neighborhood is fixed and it is the same for all of them; the communication must work reliably and synchronized.

We have showed how to model some of the most often used protocols by cellular automata. We have created a modeling tool that represents cellular automata and can help researchers to prove certain characteristics of distributed protocols.

In the near future we would like to model some other protocols as well, e.g. distributed hash tables. We are looking for possibilities to get rid of some strong limitations by finding more general cellular automata models too.

REFERENCES

[1] The network simulator. ns-2. http://www.isi.edu/nsnam/ns.

[2] Oversim: The overlay simulation framework. http://www.oversim.org.

[3] P2psim: a simulator for peer-to-peer (p2p) protocols. http://pdos.csail.mit.edu/p2psim.

[4] Peersim: A peer-to-peer simulator. http://peersim.sourceforge.net.

[5] Planetlab: An open platform for developing, deploying and accessing planetary-scale services. www.planet-lab.org.

[6] G. Balazsfalvi and J. Sztrik: Bittorrent file sharing in mobile ad-hoc networks. *Annales Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae, Sectio Computatorica*, XXVI: 159-170, 2006.

[7] A. R. Bharambe; C. Herley; and V. N. Padmanabhan: Analyzing and improving bittorrent performance. Technical Report MSR-TR-2005-03, Microsoft Research, 2005.

[8] A. R. Bharambe; V. N. Padmanabhan; and C. Herley: Analyzing and improving a bittorrent network's performance mechanisms. In *IEEE Infocom 2006*, pages 1-12, 2006.

[9] B. Cohen: Incentives build robustness in bittorrent. In: *Workshop on Economics of Peer-to-Peer Systems*, pages 251-260, 2003.

[10] J. D. Farmer; T. Toffoli; and S. Wolfram: *Cellular Automata: Proceedings of an Interdisciplinary Workshop at Los Alamos*. North-Holland, New Mexico, 1984.

[11] R. Jain: *The art of computer systems perfomance analysis*. John Wiley & Sons, Inc., New York, 1991.

[12] Leonard Kleinrock: *Queueing Systems. Volume 1: Theory.* John Wiley & Sons, Inc., 1975.

[13] S. H. Kwok: P2p searching trends: 2002-2004. *Information Processing and Management*, 42: 237-247, 2006.

[14] A. Lindenmayer and G. Rozenberg: *Automata, Languages, Development*. North-Holland, 1976.

[15] Petar Maymounkov and David Mazières: Kademlia: A peer-to-peer information system based on the xor metric. In: *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 53-65, London, UK, 2002. Springer-Verlag.

[16] S. Mullender: *Distributed systems*. Addison-Wesley, 2nd edition, 1993.

[17] J. A. Pouwelse; P. Garbacki; D. H. J. Epema; and H. J. Sips: The bittorrent p2p file-sharing system: Measurements and analysis. In: *In Proc. of the 4th International Workshop on Peer-to-Peer Systems (IPTPS '05)*, volume 3640, pages 205-216, 2005.

[18] D. Qiu and R. Srikant: Modeling and performance analysis of bittorrent-like peer-to-peer networks. In: *ACM SIGCOMM 2004*, pages 367-378, 2004.

[19] Nicola Santoro: *Design and Analysis of Distributed Algorithms*. John Wiley & Sons, Inc., 2007.

[20] A. Tanembaum and M. van Steen: *Distributed systems: Principles and paradigms*. Prentice-Hall, 2004.

[21] T. Toffoli and N. Margolus: Invertible cellular automata: a review. *Physica*, D 45:229--253, 1990.

[22] S. Wolfram: *A New Kind of Science*. Wolfram Media, 2002.

[23] G. Zihui; D. R. Figueiredo; S. Jaiswal; J. Kurose; and D. Towsley: Modeling peer-peer file sharing systems. In: *IEEE Infocom 2003*, volume 3, pages 2188-2198, 2003.