Modeling of Network Switch Controlled by Neural Network

Michal Polivka and Vladislav Skorpil

***Abstract*—Conventional switches are usually controlled by a manually defined algorithm implemented in the network processor. The conventional methods of control are fast, reliable and transparent. The article deals with alternative method of switch control – based on a neural network, focused on Quality of Service. The article describes the designed model, simulation and summarizes the results.**

***Keywords*—MATLAB, modeling, neural network, switch, switching algorithm.**

I. Introduction

Conventional switches, regardless of the category, are usually controlled by explicitly defined algorithms. The conventional solution is in most situations solid, stable and transparent. Functionality, including traffic shaping, quality of service, etc., depends on the category of determination. There are simple switches working on 2nd layer RM OSI for small and home offices and also big campus of provider switches including sophisticated functionality and usually working on the 2–4 layers RM OSI.

Most conventional switches are controlled by network processors (NPU), in some designs field-programmable gate arrays (FPGA) are used, but most of them are controlled by conventionally programmed algorithms.

The submitted solution uses for the switch control the Feedforward Backpropagation neural network. The switch was modeled using the MATLAB 2010b software.

For test purposes a simple switch with 4 input and 4 output ports was designed. The switch was designed for full duplex traffic with separate receiving and transmitting parts. The half duplex using the same transition channel is not supported in the model.

From the beginning the switch was designed as a device with Quality of Service (QoS) support.

II. Related Work

Vladislav Škorpil and Jiří Šťastný in the article "Alternatives of Converged Network Control" [1] described alternatives for the network elements control using neural networks. The article compares neural networks suitable for the network element control.

The article "Network Elements Controlled by Artificial Neural Network" [2] published on the WSEAS conference by Vladislav Škorpil and D. Novák deals with the proposal of a network controlled by neural network. The presented network element and model are different from the switch proposed in this article.

Vladislav Škorpil and S. Kamba present in the conference paper "Back Propagation and Genetic Algorithms for Control of the Network Element" [3] Backpropagation neural networks.

Vladislav Škorpil and Michal Polívka in the article "Modeling Logical Function Antivalence Using Neural Network in MATLAB" [4] described in detail the Feedforward Backpropagation neural network and neural network simulation in the MATLAB software.

III. Model

As was mentioned in the section I, the modeled switch has 4 input and output ports. The count of ports was chosen to be easy to model and, at the same time, to show all errors in a proposal design.

The main block diagram of the switch is in Fig. 1. The main blocks are – packet generator, input queues, switch fabric (with neural network control blocks) and output queues. The output queues are the "Output FIFO" blocks in Fig. 1.

The model was created above all in the Simulink. The control neural network was designed in MATLAB neural network toolbox (without using "nntool") and converged to the Simulink using "gensim" tool, for reasons see [4]. Some error check scripts are also developed in the plain MATLAB.

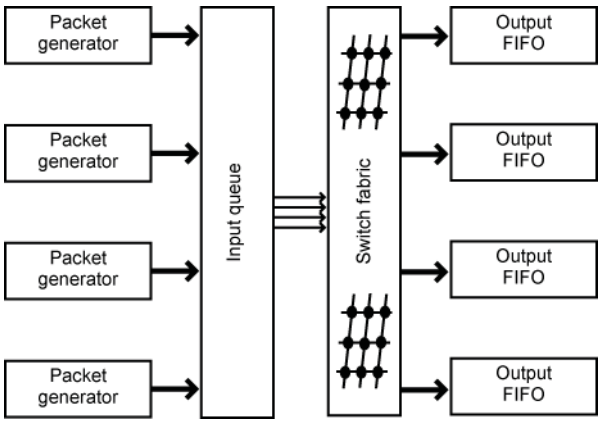

Fig. 1: Base switch schema

Manuscript received April 11, 2012. This research was supported in part by the grants: No. ED 2.1.00/03.0072, Centre of sensor information and communication systems (SIX), CZ.1.07/2.2.00/28.0062, Joint activities of BUT and TUO while creating the content of accredited technical courses in ICT and No. FEKT-S-11-15 Research of electronic communication systems.

M. Polivka is with the Department of Telecommunications, Faculty of Electrical Engineering and Communication, Brno University of Technology, Purkyňova 118, Brno, 612 00 Czech Republic (e-mail: michal.polivka@phd.feec.vutbr.cz).

V. Skorpil is with the Department of Telecommunication, Brno University of Technology, Purkyňova 118, 612 00 Brno, Czech Republic, phone: +420 541 149 212, e-mail skorpil@feec.vutbr.cz.

A. Simulation Setup

In the 1st RM OSI layer in the Ethernet network the transmitted signal is continual and changing in time. To make the simulation transparent, the simulation works on low layers with continual time too.

The Simulink uses the Solver with "Fixed-step" and discrete ("no continuous states"). It is because there is no requirement to study the transitional effects. The "Fixed-step size" selected was 0.1 as a compromise between accuracy and simulation speed.

The simulation starts at time 0 and stop at time 20, but the length of simulation is modified in dependence on the kind of experiment.

In some special situations "Real-time" module is used, which is based on the "waitforreal" script made by Stan Quinn and presented by MathWorks Inc. [5]. The script and Simulink module are useful in cases the "Real-Time Windows Target" module is unavailable (or uselessly complicated).

The whole model is based only on the basic Simulink blocks, a few "Signal Processing Blocksets" and "Neural Network Toolbox".

B. Packer Generator

The model contains 4 identical packet generators. They are synchronized and initialized by a special "Starter" block. The "Starter" block is user configurable before the simulation starts. The "Starter" can be configured separately for each from the 4 generator blocks. The packet generation can be started in a random period, or in an exact period defined by the user.

A situation similar to generator synchronization is in the core of the packet generator. The packet generator can generate all the generated pars randomly or user-defined.

From the model point of view in case the packet generator does not emit the packet, it emits the 0 signal.

For the testing purposes a simplified packet was designed – it contains only the label (number) of source port (which port the packet comes from), destination port label, the array of packet priority and the array of data. The packet is visualized in Fig. 2

The modeled packet consists of:

- The "source port" is number 1–4.
- The "destination port" is number 1–4.
- The "priority" array is put together from 2 parts (and 2 numbers). Efforts were made to simulate the DSCP (Differentiated Services Code Point) array [6], [7], [8], [9]. In the model the first number is from the interval 1–4 (indicates priority), the higher number means higher precedence. The second number is from interval 1–3 and indicates the probability of packet drop (when it is necessary). In the case of drop probability the higher number means a higher probability of packet drop. Priority calculation is given by equation (1), where P determines packet precedence and d determines drop probability. Both numbers – P and d – are integer numbers and can be random or user-defined.

$$priority = P \cdot 10 + d \tag{1}$$

- The last array "Data" is also a whole number (integer) and represents the real transmitted data – in binary numbers the array "Data" is of variable length.

0	1	2	3	4	5
Source Port	Destination Port	Priority		Data	

Fig. 2: Packet model

Because the source and destination ports can be randomly selected, it is important to check if any source and destination ports are the same. This check follows after packet generation. Packets with the same source and destination ports are dropped.

The "Data" array has a secondary purpose as packet identifier. Because in "standard" random mode it is a number from the interval 1–10,000 there is low probability of packet repetition with the same array data and source and destination ports.

The real Ethernet traffic is from the switch point of view usually pseudorandom – it means that the packet can be received in pseudorandom time. This is the reason why the generated packets are stored in the FIFO (First In, First Out) queues, till they are sent to the packet generator output in random or user-defined time – its simulate jitter.

C. Input Queue

The "input queue" block offers on the top level the user a choice from 2 possibilities of packet processing in the case of destination collision. If the block does not indicate packet collision, the packets are immediately transferred to the switch fabric and to the destination.

Choice 1 in the case of collision drops a packet with the lower precedence P. If more than one packet are in collision, the packet with best precedence array will be transferred and others will be dropped.

Choice 2 is much more sophisticated in comparison with choice 1. In this case the effort is not to drop the packets and fully support the QoS [9].

Two or more packets with detected destination collision are transferred to the QoS block, which divided the packets into groups. For each output port one FIFO queue is defined for 8 packets – it means if there are more than 8 packets in the queue, the new packets are dropped. This algorithm is known as "tail drop" with "aggressive dropping" [9], [10]. For each precedence 1 queue is modeled.

The queuing mechanism is in the model represented by 1 block, which is easy to replace or modify. The queuing algorithm is based on the "Priority Queuing" (PQ) algorithm or rather on the newer "Low Latency Queuing" (LLQ) algorithm [9]. For the model purpose is the LLQ algorithm simplified – the algorithm do not protect a queues with lower precedence. In the LLQ terminology, the packet with highest available precedence is sent as packet with "strict-priority".

In case the output interface (port) is empty and ready for packet reception, the packets from queues are delivered. The algorithm starts by sending packets from the queue with a higher precedence. In case the queue is empty, the algorithm continues to the queue with a lower precedence, etc.

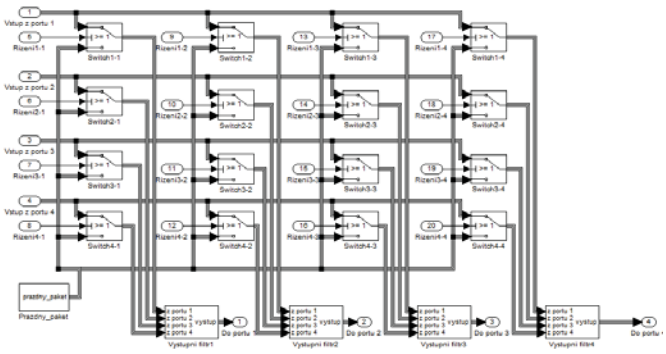

Fig. 3: Switch fabric

D. Switch Fabric

The switch fabric block consists of two main parts – configuration matrix generator and switch (cross) fabric.

The switch fabric design is shown in Fig. 3. The switch fabric is designed as a 4×4 array of switches.

The switch fabric is controlled by configuration matrix c, which is generated by the control matrix, for example see (2).

$$c = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{2}$$

The designed switch works fully in parallel; a consequence of this solution is, that the switch has the same performance irrespective of the number of ports (in case of ignore a packet collision). This is illustrated in Fig. 4, the blue boxes represent 4 identical neural networks (one for each port). The other blocks in Fig. 4 transpose the neural network output vectors, round the vectors and concentrate the vectors into one configuration matrix.

The neural network input vector p is comprised of 2 labels (numbers) – source and destination ports. The output (target) vector t consists of 4 binary numbers. For example see (3).

$$p = \begin{pmatrix} 4 \\ 3 \end{pmatrix}; t = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \tag{3}$$

In section I it was mentioned that the switch control is based on the Feedforward Backpropagation neural network.

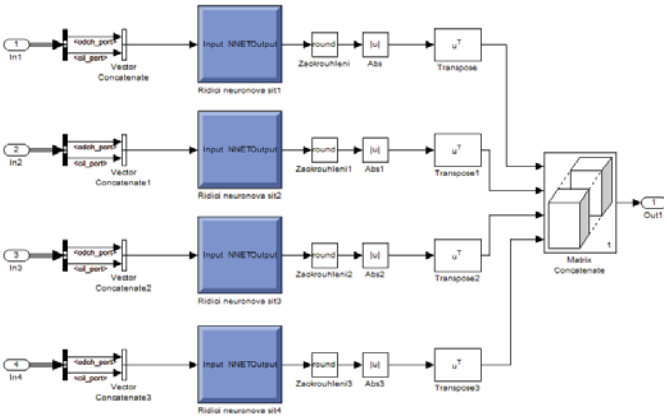

Fig. 4: Configuration matrix generator

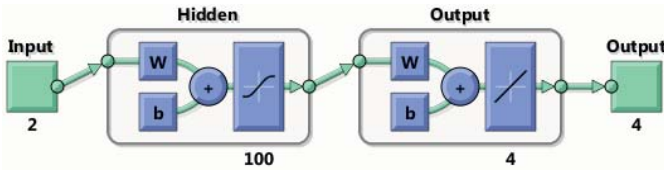

Fig. 5: Control neural network visualization

The network is recommended by [11] as the first choice for testing in this kind of tasks. The choice was confirmed by the results reached in [4] and also during fitting the network submitted solution. In the hidden layer of network 100 neurons have been used, the output layer consists of 4 neurons, see Fig. 5.

The example part of the training set follows.

```
p = [4 3 2 1, 4 3 2 1, …,4 4 4 3…;
     3 2 1 4, 2 1 4 3, …,1 2 3 1…];

t = [0 0 1 0, 0 1 0 0, …,1 0 0 1…;
     0 1 0 0, 1 0 0 0, …,0 1 0 0…;
     1 0 0 0, 0 0 0 1, …,0 0 1 0…;
     0 0 0 1, 0 0 1 0, …,0 0 0 0…];
```

The training set was divided:

- train ratio uses 90 % of the set
- validation ratio 5 %
- test ratio also 5 %

The network quickly converged to the desired mean square error (mse) [11]. After the 3rd epoch the mse reaches $mse = 4.306 \cdot 10^{-21}$. The fitting is shown in Fig. 6.

E. Output FIFO queue

To each output port one FIFO queue is dedicated. The queue size is set to 8 (as in the case of input FIFO). The packets are stored in output FIFO and read only in case the destination port on the opposite device is ready for packet reception.

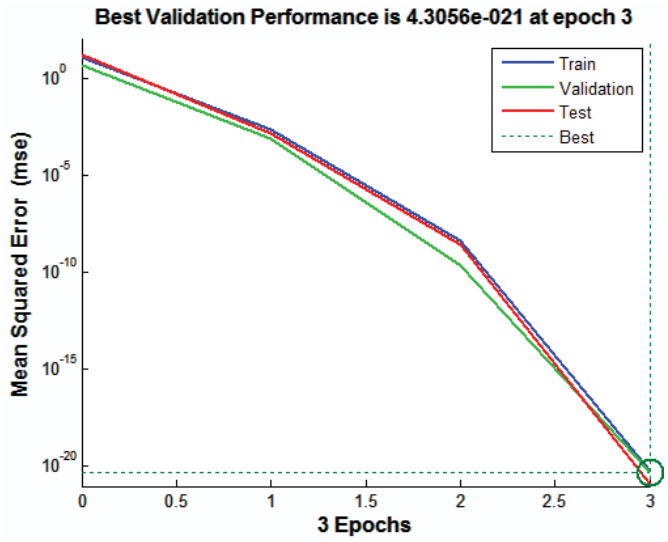

Fig. 6: Neural network performance (fitting)

The output FIFO queues work with input queues as dejitter buffer [9], [12], [13]. In the model this is simulated by the random number generator.

IV. Results and Testing

The designed model is equipped with test and visualization tools to validate the simulation results. The main validation tool is the "comparison table", see Fig. 7. In Fig. 7 there are 2 parts distinguished by colors – the green part is a packet sent by the packet generators to the switch (G1–4) i.e. "Generator". The red part of the comparison table and Fig. 7 identify the packets after the switching process at the output FIFO (C1–4).

The two cells (in Fig. 7 are both marked by red square) are one packet (structure is described in section III.B and in Fig. 2). The packet is highlighted at the generator output and again at the switch output. The lines of cells correspond after transposition with Fig. 2.

- first row – output port
- second row – destination port
- third row – priority
- forth row – data

The horizontal axis is the time axis. Fig. 7 represents 12 steps. The packet generated in simulation step 2 is at step 3 sent out from the switch.

The zeros in some cells indicate "no packet" – no one packet is generated.

Some packets are received after more than one simulation step, which is a consequence of the active QoS and packet collisions. It is marked by blue color in Fig. 7. The two packets with the same destination port are sent from generator 1 and also 2. The packet from generator 2 has priority 42, i.e. higher than the packet from generator 1 with priority 22. This is a consequence of the packet sent from generator 2 being delivered before the packet sent from generator 1.

G1	0	0	0	0	1	1	0	0	1	1	0	0
	0	0	0	0	2	2	0	0	3	4	0	0
	0	0	0	0	32	42	0	0	22	11	0	0
	0	0	0	0	1311	8716	0	0	931	9036	0	0
G2	0	2	0	2	0	0	0	2	2	0	0	0
	0	3	0	3	0	0	0	1	3	0	0	0
	0	23	0	11	0	0	0	23	42	0	0	0
	0	3701	0	1506	0	0	0	9126	4431	0	0	0
G3	0	3	0	0	0	0	3	0	3	3	0	0
	0	4	0	0	0	0	1	0	4	2	0	0
	0	41	0	0	0	0	22	0	23	12	0	0
	0	1401	0	0	0	0	3421	0	7831	4036	0	0
G4	0	4	0	0	0	0	4	4	0	4	0	4
	0	3	0	0	0	0	3	2	0	2	0	3
	0	33	0	0	0	0	33	33	0	43	0	42
	0	5201	0	0	0	0	4721	3726	0	2936	0	6746
C1	0	0	0	0	0	0	3	2	0	0	0	0
	0	0	0	0	0	0	1	1	0	0	0	0
	0	0	0	0	0	0	22	23	0	0	0	0
	0	0	0	0	0	0	3421	9126	0	0	0	0
C2	0	0	0	0	1	1	0	4	0	4	3	0
	0	0	0	0	2	2	0	2	0	2	2	0
	0	0	0	0	32	42	0	33	0	43	12	0
	0	0	0	0	1311	8716	0	3726	0	2936	4036	0
C3	0	4	2	2	2	0	4	0	2	2	1	4
	0	3	3	3	3	0	3	0	3	3	3	3
	0	33	23	11	23	0	33	0	42	23	22	42
	0	5201	3701	1506	3701	0	4721	0	4431	3701	931	6746
C4	0	3	0	0	0	0	0	0	3	1	0	0
	0	4	0	0	0	0	0	0	4	4	0	0
	0	41	0	0	0	0	0	0	23	11	0	0
	0	1401	0	0	0	0	0	0	7831	9036	0	0

Fig. 7: Comparison table – 12 simulation steps

V. Conclusion

The submitted model made in the MATLAB, validated the hypothesis about the possibility of controlling switches using the neural network.

The presented switch is different from those proposed in [2] or [3] and uses a different switching strategy in comparison with standard commercial switches.

For the future there is a strategy for implementing the designed switch and algorithm in hardware. The core of the model – neural network will be converted to the VHDL, using the MATLAB HDL Coder, and the code will be used in the NetFPGA system [14].

The hardware implementation allows comparing the actual performance of the modeled switch with that of other switches.

References

[1] V. Skorpil and J. Stastny, "Alternatives of Converged Network Control," *Elektrorevue*, vol. 1, pp. 19-23, Dec. 2010.

[2] V. Skorpil and D. Novák, "Network Elements Controlled by Artificial Neural Network," in *Proceedings of 14th WSEAS International Conference on Communications*, Corfu Island, Greece, 2010, pp. 145-148.

[3] V. Skorpil and S. Kamba, "Back Propagation and Genetic Algorithms for Control of the Network Element," in *In 34th International Conference on Telecommunications and Signal Processing*, Budapest, Hungary, 2011, pp. 240-243.

[4] M. Polivka and V. Skorpil, "Modeling Logical Function Antivalence Using Neural Network in MATLAB," in *13th International Conference on Research in Telecommunication Technologies RTT - 2011*, Brno, 2011, pp. 1-3.

[5] The MathWorks, Inc. (2010, July) How do I get a real time clock in simulations using Simulink? [Online]. http://www.mathworks.com/support/solutions/en/data/1-15JAW/

[6] S. Blake et al. (1998, Dec.) RFC 2475: An Architecture for Differentiated Services. [Online]. http://datatracker.ietf.org/doc/rfc2475/

[7] K. Ramakrishnan, S. Floyd, and D. Black. (2001, Sep.) RFC 3168: The Addition of Explicit Congestion Notification (ECN) to IP. [Online]. http://datatracker.ietf.org/doc/rfc3168/

[8] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. (1999, June) RFC 2597: Assured Forwarding PHB Group. [Online]. http://datatracker.ietf.org/doc/rfc2597/

[9] W. Odom, R. Healy, and N. Mehta, *Směrování a přepínání sítí – Autorizovaný výukový průvodce*. Brno: Computer Press, a.s.; Cisco System, Inc., 2009.

[10] W. Odom and M. J. Cavanaugh, *Cisco QoS Exam Certification Guide (IP Telephony Self-Study), 2nd Edition*. Indianapolis: Cisco Press, 2004.

[11] M. H. Beale, M. T. Hagan, and H. B. Demuth, *Neural Network Toolbox 7*. Natic: The MathWorks, Inc., 2010.

[12] J. H. Chao and B. Liu, *High Performance Switches and Routers*. New Jersey: JohnWiley & Sons, Inc., 2007.

[13] I. Elhanany and M. Hamdi, *High-performance Packet Switching Architectures*. London: Springer, 2007.

[14] NetFPGA. (2012) Netfpga.org. [Online]. http://www.netfpga.org/