Theoretical and Applied Informatics
ISSN 1896–5334
Vol. 25 (2013), no. 1
pp. 49–66
DOI: 10.2478/thai-2013-0005

Analysis of CHOKe-Family Active Queue Management

Adam Domański[1], Joanna Domańska[2], Jerzy Klamka[2]

[1] Institute of Informatics
Silesian Technical University
Akademicka 16, 44–100 Gliwice, Poland
adamd@polsl.pl
[2] Institute of Theoretical and Applied Informatics
Polish Academy of Sciences
Baltycka 5, 44–100 Gliwice, Poland
{joanna,jklamka}@iitis.gliwice.pl

Received 1 February 2013, Revised 10 April 2012, Accepted 10 May 2013

Abstract: In the article we study a model of network transmissions with Active Queue Management in an intermediate IP router. We use the OMNET++ discrete event simulator to model the varies variants of the CHOKe algoithms. We model a system where CHOKe, xCHOKe and gCHOKe are the AQM policy. The obtained results shows the behaviour of these algorithms. The paper presents also the implementation of AQM mechanisms in the router based on Linux.

Keywords: Computer Networks, Active Queue Management, CHOKe algorithm

1. Introduction

In order to meet the demands of providing high quality services, IETF proposes to implement in routers conguestion detection mechanisms and reacting appropriately when they occur. The best known mechanism of this type is Random Early Detection algorithm [3]. For this mechanism packet can be can be dropped from queue if there is a risk of the router overloading. This solution has many advantages inter alia RED prevents global synchronization of TCP sources. This effect is achieved by co-operation between the RED and congestion control mechanisms built into TCP transmitters (eg. New Reno). When the packet is droped the TCP source decreased the transmission speed. This mechanism does not work well with streams without conqestion control and for TCP streams which does not control congestion basing on packet loss (eg. Vegas). These mechanisms when competing for bandwidth with the New

Reno transmitters behaves more aggressively. This phenomenon has resulted in AQM algorithms, distinguishing data streams. For this algorithms the probability of droping the packet from queue is proportional to the number of packets from this stream in the bufor. In this article we present the five variants of CHOKe algoritm. Using OMNET++ discrette event simulator we present the behavior of these mechanisms.

The rest of this article is organized as follows. The section 2 describes two variants of CHOKe algorithms. The section 3 presents proposed simulation models and shows obtained results. The section 4 discusses the implementation of CHOKe/ AQM algorithms in Linux. The conclusions are presented in section 5.

2. The CHOKe algorithms

CHOKe is stateless AQM algorithm similar to RED used incoming packages to punish streams with the highest demand for bandwidth [17].

Just as in the case of RED, there are two threshold values: Min_{th} and max_{th}. When a new package arives the new average queue length is calculated. When the average queue length is less than Min_{th}, all packets are placed in the buffer. When the average queue length is greater than Min_{th}, CHOKe pulls randomly one packet from the FIFO buffer (”CHOKe victim”) and verifies if the package comes from the same stream as an incoming packet. If the both packets belong to the same stream, both are removed (this situation id called ”CHOKe hit”). Otherwise, the randomly selected package (”CHOKe victim”) is returned to the buffer and the arived packet is placed in the queue with P probability. This probability is calculated in the same manner as in the case of RED algorithm. This event is called the ”choke miss”. The figure 1 presents the CHOKe algorithm. The Front CHOKe algorithm introduces modifications to the algorithm CHOKe. This algorithm does not drow a packet. The first packet in the queue is a ”CHOKe victim” [17]. Another very similar modification of CHOKe algorithm is Back CHOKe. Its principle of operation is obvious [17]. This modification takes the last packet from the queue and compare its with incoming packet (figure 2).

Algorithm xCHOKe remembers ”CHOKe hit” events in the special table called ”lookup table” [18]. Algorithm starts to work when the queue length is between Min_{th} and Max_{th}. For all incoming packets algorithm scan ”lookup table” to find flow id identical to the flow id of incoming packet. If the search is successful (”table hit”) the incoming packet is dropped with p^* probability. Next, arrival packet is compared to packet drown from queue (”CHOKe victim”). if both have the same stream id, both are removed and xCHOKe scan ”lookup table” again. I looks for id of flow of dropped packets and increment ”hit counter” parameter assocciated with flow id of dropped packet. If the search is unsuccesfull algorithm creates a new row in the table with ”hit counter” equal to one (figure 3).

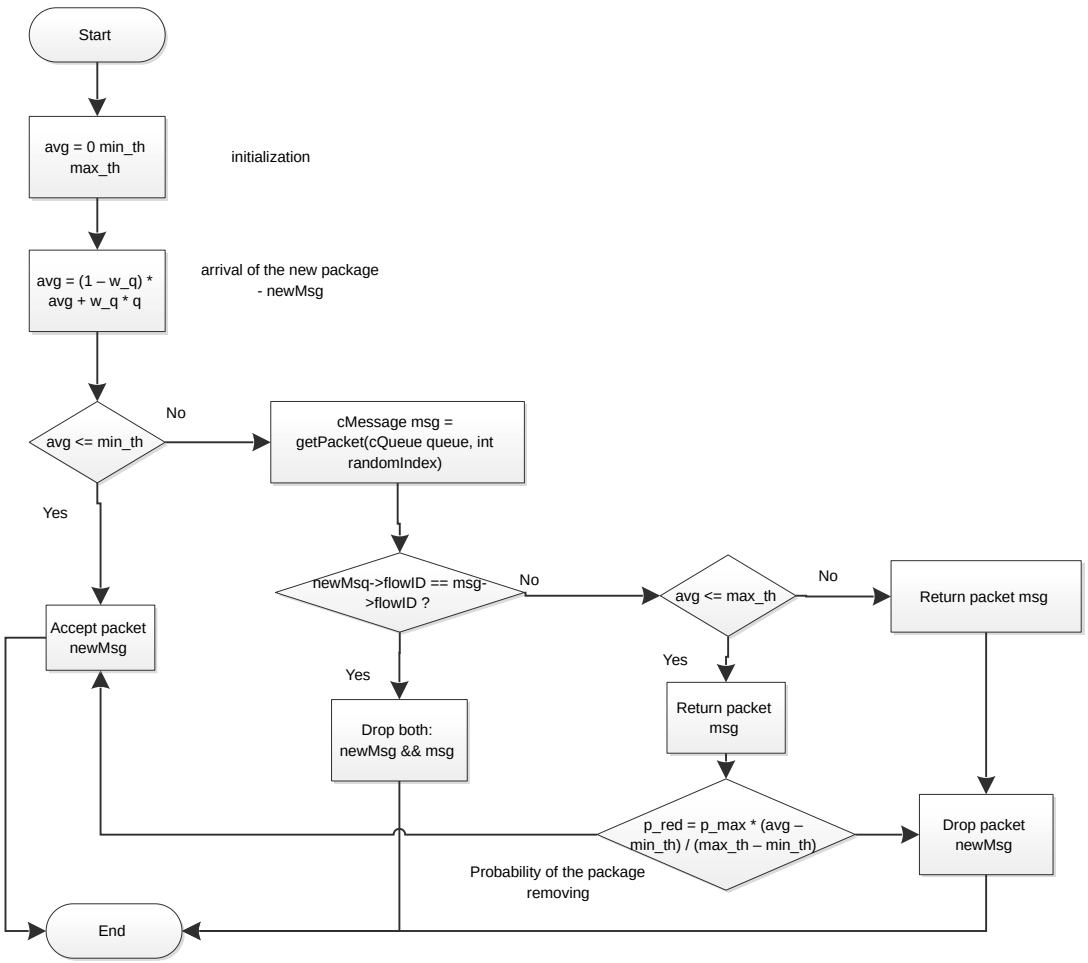

Fig. 1. Diagram of the CHOKe algorithm

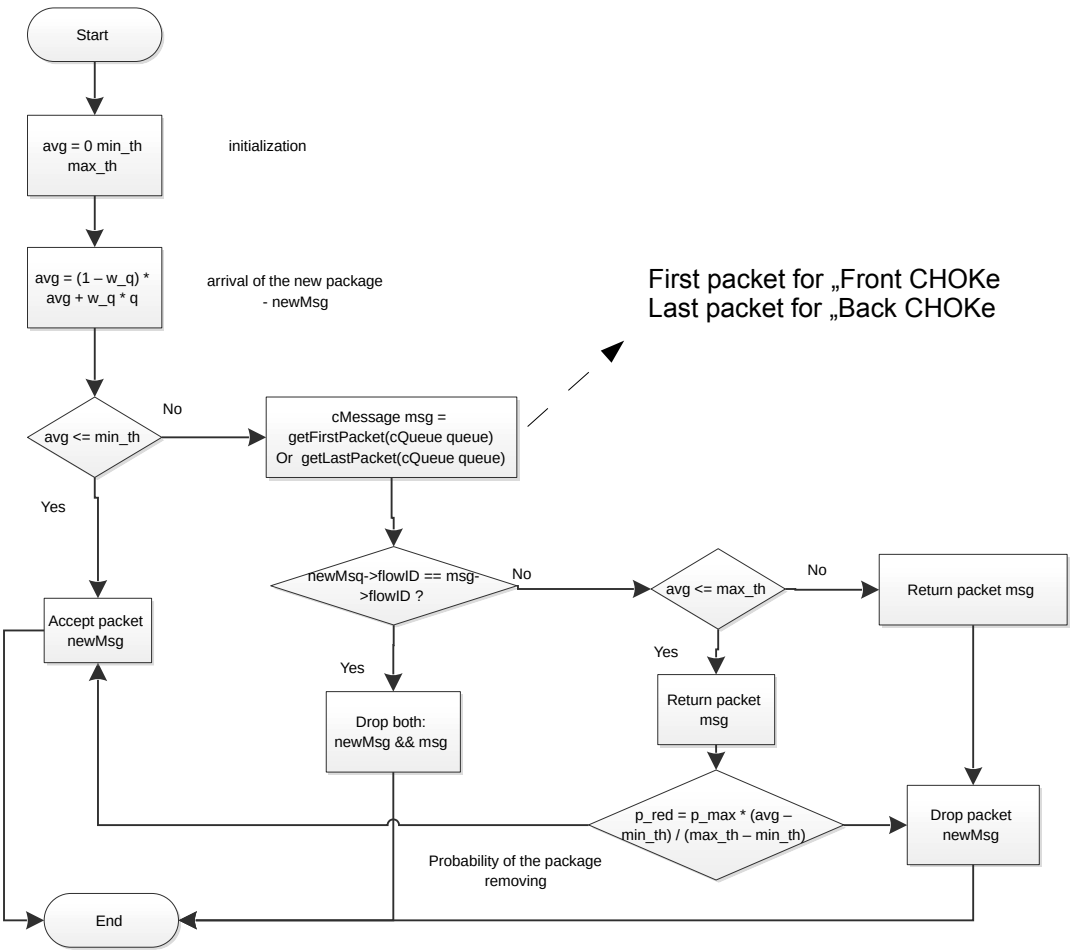

Fig. 2. Diagram of the Front (Back) CHOKe algorithm

Probability of dropping packet table hit is $p^* = MIN(1, p_{RED} * 2^n$, where p_{RED} is prabability calculating by RED algorithm, n is ”hit counter” for searched flow id.

Geometric Choke (gCHOKe) is a modification of the CHOKe algorithm proposed by Addisu Eshete i Yuming Jiang from Norwegian University of Science and Technology in Trondheim [10]. This algorithm has the additional, configurable parameter *maxcomp* $\in [1..\infty)$. This parameter determines the maximum number of successful comparisons. The algorithm compares the incoming packet with a random packet from the queue (”CHOKe Victim”). The comparison is successful when both packages are from the same stream. Comparison ends when: the comparison is unsuccesfull or the number of comparisons is *maxcomp*. In this case, all matching packets, and the incoming are removed from the queue. If the first match is not successful, the random packet (”CHOKE Victim”) comes back to the queue. The arrival packet is placed in the queue with *P* probability (figure 4). The CHOKe algorithm is a special case of gCHOKe for *maxcomp* = 1.

3. Simulation results

Research has been conducted in the OMNET++ simulation environment of discrete events. To emphasize the importance of using self-similar sources of traffic the comparative research has been carried out for the self-similar and poisson source. Input traffic intensity was chosen as $\alpha = 0.5$ or $\alpha = 0.1$, and due to the modulator characteristics, the Hurst parameter of self-similar traffic was fixed to $H = 0.8$. For both considered in comparisons cases, i.e. for geometric interarrival time distribution (which corresponds to Poisson traffic in case of continuous time models) and self-similar traffic, the considered traffic intensities are the same. A detailed discussion of the choice of model parameters is also presented in [7].

The parameters of AQM buffer:

- $Min_{th} = 100$,
- $Max_{th} = 150$,
- buffer size (measured in packets) = 200,
- weight parameter $\alpha = 0.007$.

For parameter $\alpha = 0.5$ the traffic generated by a single source was too large. For experiments with a single source queue occupancy fast exceeds the parameter Max_{th} and ”CHOKe-victim” always was dropped. So the rest part of the article describes results obtained for $\alpha = 0.1$.

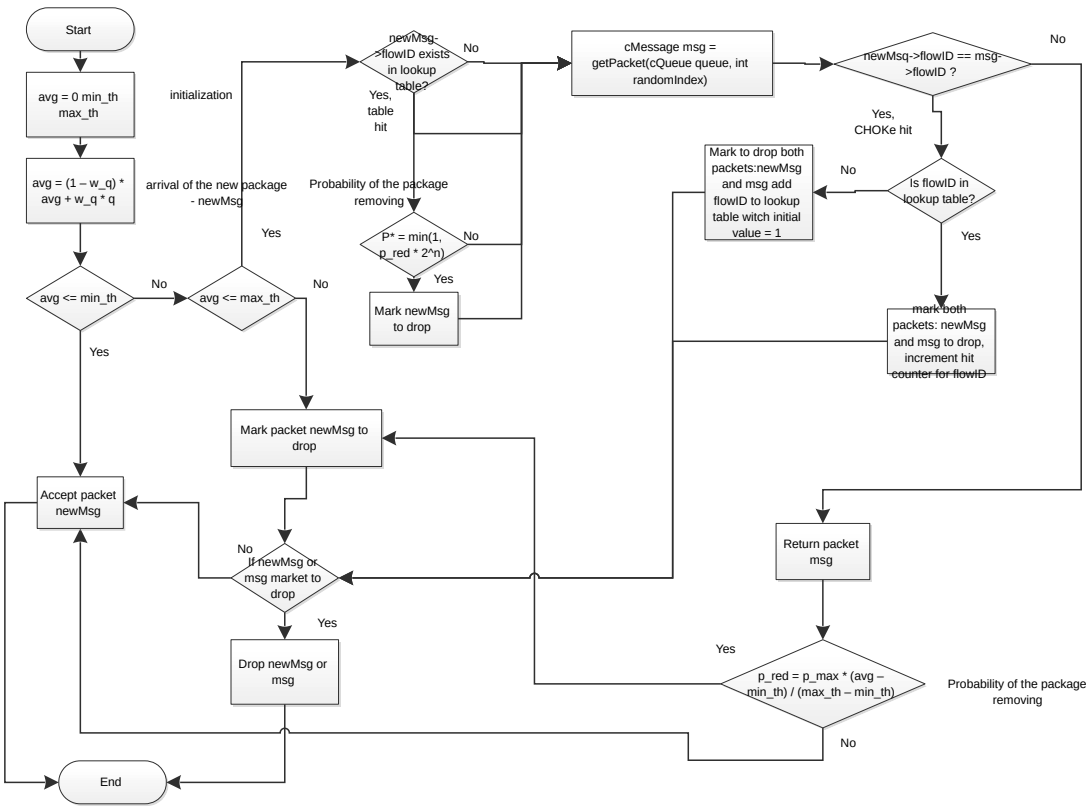

Fig. 3. Diagram of the xCHOKe algorithm

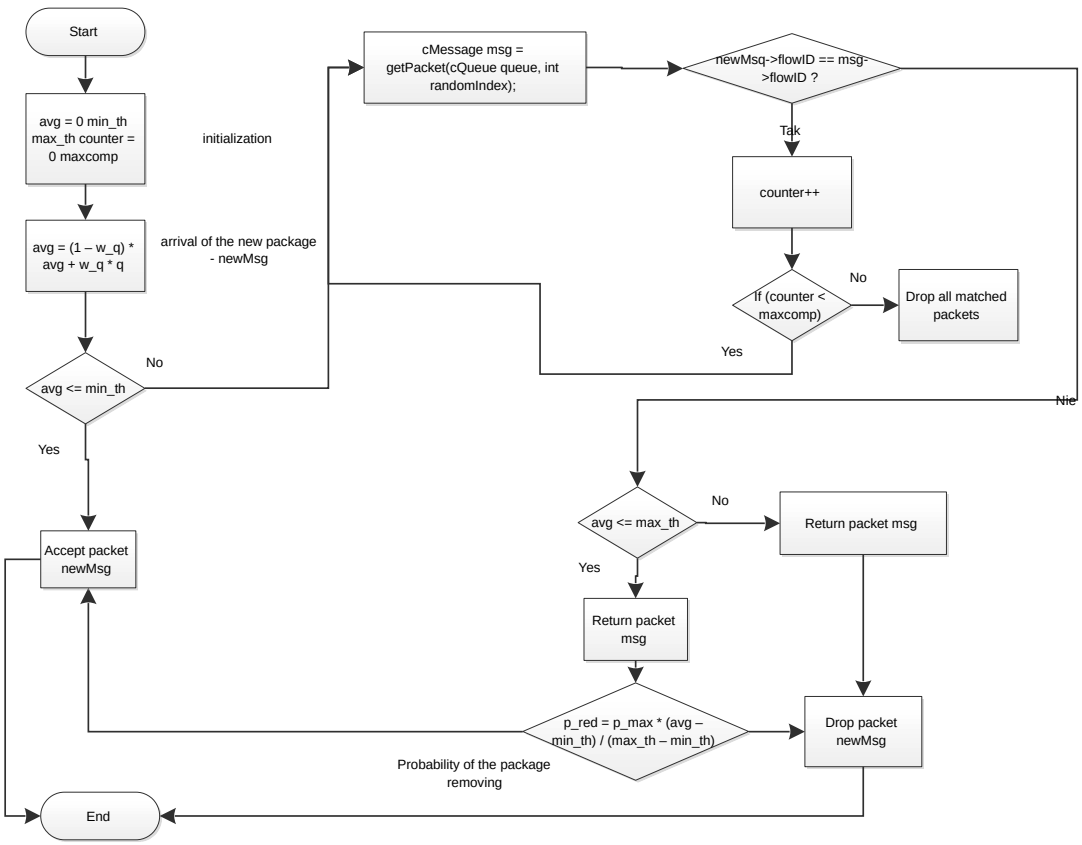

Fig. 4. Diagram of the gCHOKe algorithm

The results are presented in tab. 1. The obtained results were grouped according to the selected CHOKe algorithm, type of sorce (geometrical or selfsimilar) and number of sources.

As can be seen for 5 geometrical sources (regerdless of type od CHOKe) we received a zero number of dropped packets and a small waiting time. This situation is caused by small queue load (see figure 5). Average queue length never exceeds Min_{th} value. The situation is quite different for selfsimilar sources (figure 6). This is due to the fact that the selfsimilar sources (with the same parameters as the geometrical sources) generate more traffic (these results confirmed previous studies [7][8]). The queue exceeds the Min_{th} size. Packages are destroyed by the CHOKe and RED mechanisms. However, the queue occupancy distribution is very uneven (figure 6 (left)). When the number of streams increases the number of packets discarded by CHOKe decreases (decreases the probability of selecting ”CHOKE victime” from the same source). Interestingly the average waiting time does not increase significantly (figure 6 (right), figure 9(left)). What proves the thesis that the choke is suitable for aggressive streams management. The streams generating a larger number of data lose more packets in the overloading buffer. The above-mentioned arguments are true only for self-similar flows. The results can depend on the statistical distribution of packets in the buffer. This thesis proves the extremely large number of packets discarded by the Back CHOKe mechanism. Algorithms such xCHOKe and gCHOKe can drop more packets in one step of algorithm. What causes an increase in the number of lost packets. The queue occupancy and the average waiting time distributions are unregular (figures 7, 8, 9).

Source	Nb of sources	Algorithm	Droped by RED	Droped by CHOKe	Avg waiting time
geo	5	CHOKe	0	0	6.56175
geo	10	CHOKe	78827	678960	240.423
geo	15	CHOKe	201455	779864	374.484
self	5	CHOKe	379747	1788726	170.6
self	10	CHOKe	2635	480420	171.703
self	15	CHOKe	1835	318191	172.104
geo	5	fCHOKe	0	0	6.56175
geo	15	fCHOKe	202114	777111	353.554
self	5	fCHOKe	15872	420695	174.505
self	15	fCHOKe	16317	318318	190.382
geo	5	bCHOKe	0	0	6.56175

Source	Nb of sources	Algorithm	Droped by RED	Droped by CHOKe	Avg waiting time
geo	15	bCHOKe	186	2644617	191.701
self	5	bCHOKe	10	1950025	124.281
self	15	bCHOKe	6	648573	101.832
geo	5	xCHOKe	0	0	6.56175
geo	15	xCHOKe	208	2644061	162.98
self	5	xCHOKe	12	1950320	81.9792
self	15	xCHOKe	28	637522	113.098
geo	5	gCHOKe	0	0	6.56175
geo	15	gCHOKe	1201	423	265.371
self	5	gCHOKe	4095300	300788	90.5272
self	15	gCHOKe	611768	35950	85.63

Tab. 1. The obtained results: number of dropped packets and average waiting times

4. Linux implementation

This chapter presents the details of the CHOKe algorithms implementation in the real software router based on the Linux. The main objective of this stage of the study has been the confirmation of simulation results in the real working network. This chapter provides a description of the created program and obtained results.

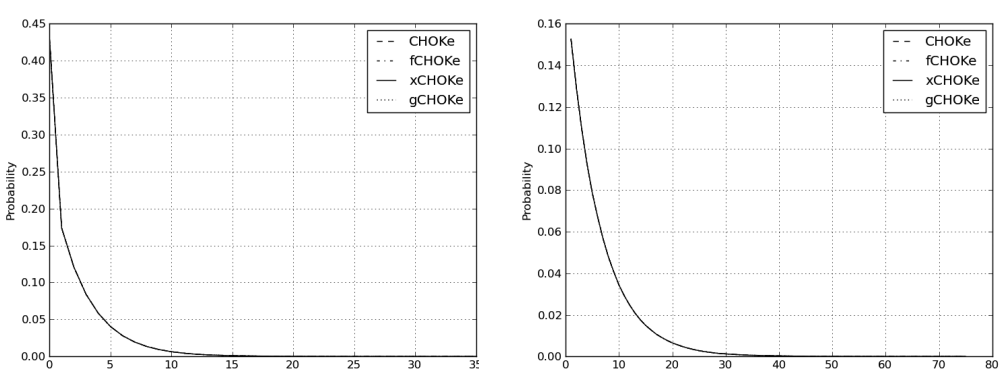

Fig. 5. Queue length distribution (left) and Waiting times distribution (right), CHOKE, Front CHOKe, gCHOKe, xCHOKe queues, geometric source, $\alpha = 0.1$, $\mu = 0.25$, $w = 0.007$, 5 sources

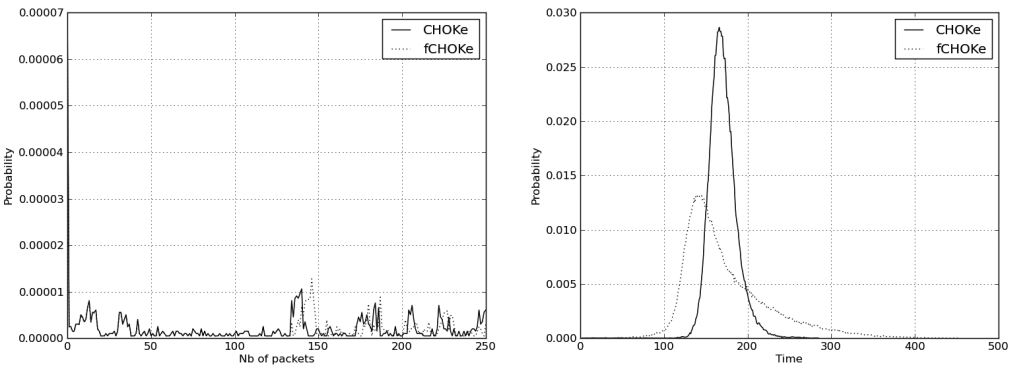

Fig. 6. Queue length (left) and Waiting times distribution (right),
CHOKE, fCHOKe queues, selfsimilar source,
$\alpha = 0.1$, $\mu = 0.25$, $w = 0.007$, 5 sources

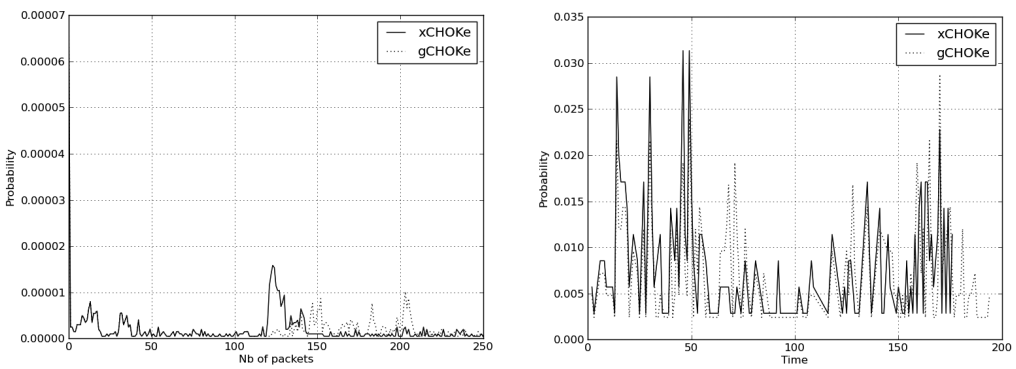

Fig. 7. Queue length (left) and Waiting times distribution (right),
xCHOKE, gCHOKe queues, selfsimilar source,
$\alpha = 0.1$, $\mu = 0.25$, $w = 0.007$, 5 sources

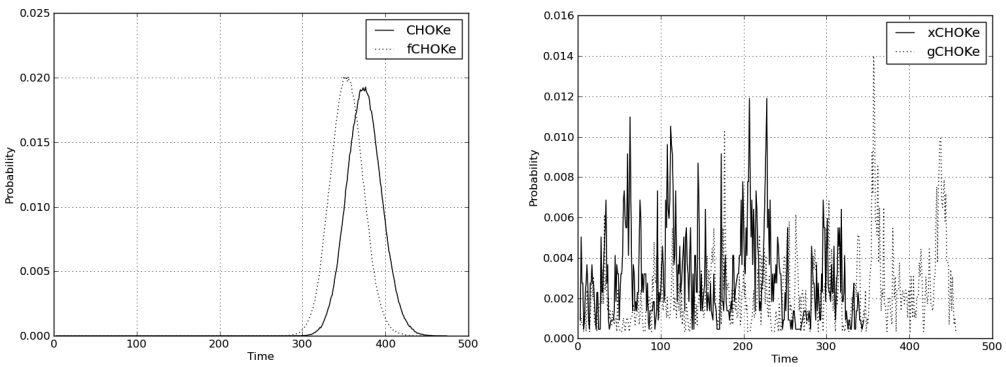

Fig. 8. Waiting times distribution CHOKe, fCHOKe (left), xCHOKE, gCHOKe (right) queues, geometric source, $\alpha = 0.1$, $\mu = 0.25$, $w = 0.007$, 15 sources

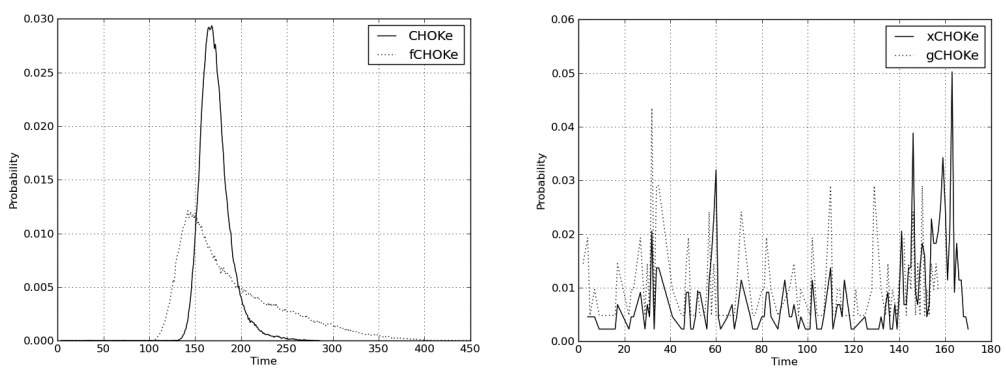

Fig. 9. Waiting times distribution CHOKe, fCHOKe (left), xCHOKE, gCHOKe (right) queues, selfsimilar source, $\alpha = 0.1$, $\mu = 0.25$, $w = 0.007$, 15 sources

The implementation was based on the Iptables – a fundamental tool to manage data streams in Linux. The total packets flow in the iptables is shown in figure 10.

Iptables is used to redirect all traffic to the user queue supported by the application (in user space). This support has been created using netfilter framework. The netfilter library performs the following tasks:

- connection to iptables,
- getting packets from the queue,
- send information to iptables about decision of throwing packet.

The entire program is divided into two threads synchronized by a special semaphore with a maximum size of one, called mutex. Code of the algorithm is as follows:

```
while true:
wait for packet in buffer:
process the packet
```

Diagram of the program is shown in figure 11. The configuration of the iptables is as follows:

```
iptables -I FORWARD 1 -j QUEUE
```

This configuration is equal to:

```
iptables -I FORWARD 1 -j NFQUEUE --queue-num # traffic is directed to the buffer 0.
```

Execution of the program is as follows:

```
./aqmrun [-m size] [-M size] [-Q number] [-w weight] [-b size] [-c number] [r] [f] [b] [
```

where:

- m – min_{th}
- M – max_{th}
- Q – netfilter queue number (default 0 - this parameter depends on the Iptables configuration)
- w – weight parameter α (default 0.002)
- b – buffer size (default 2000)
- c – *maxcomp* parameter for gCHOKe (default 1)
- r – standard CHOKe (default)
- f – Front CHOKe
- b – Back CHOKe
- g – Geometric CHOKe

Figure 12 displays the experiment topology of the network. The computer between two laptops is the most important part of the network. It works as a router with AQM algorithms implemented inside. The router running the Linux operating system and its main task is forwarding packets between two laptops. One of them is connected to access point 802.11b. This is old type of AP and works with speed 11 Mb/s network. This connection is the bottleneck. The second laptop is connected to 100 Mb/s Ethernet. During the test the laptop connected to the Ethernet link transmits data to the second laptop and the behaviour of the AQM algorithms is observed.

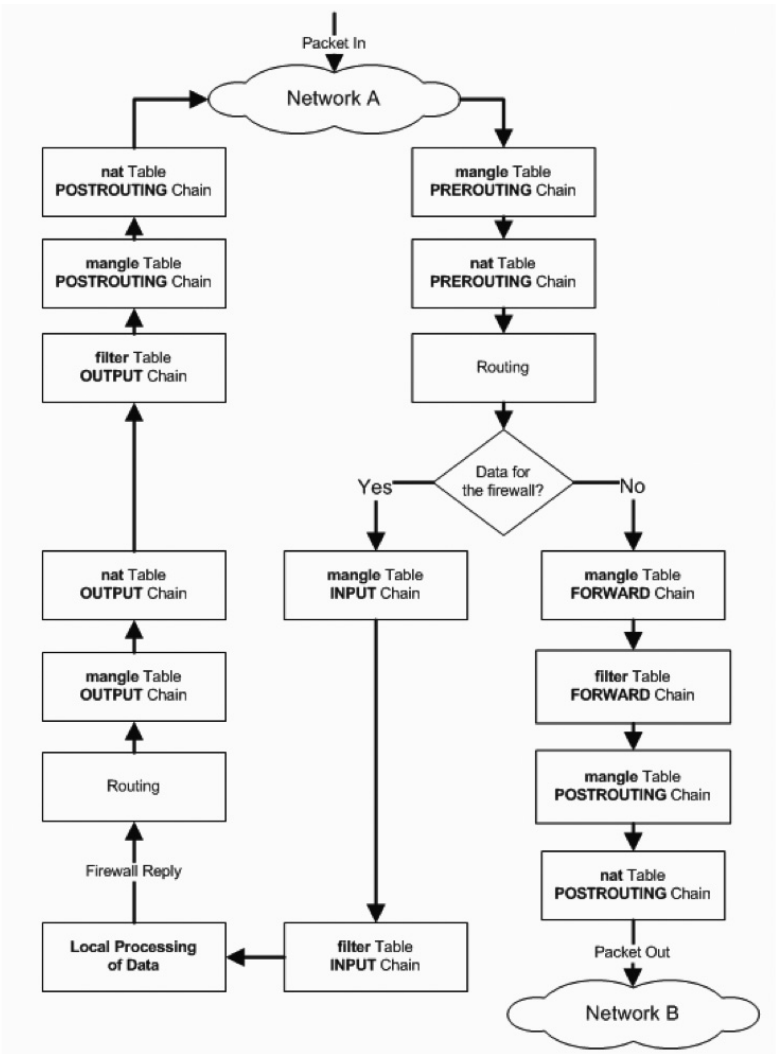

Fig. 10. The packets movement through the Iptables

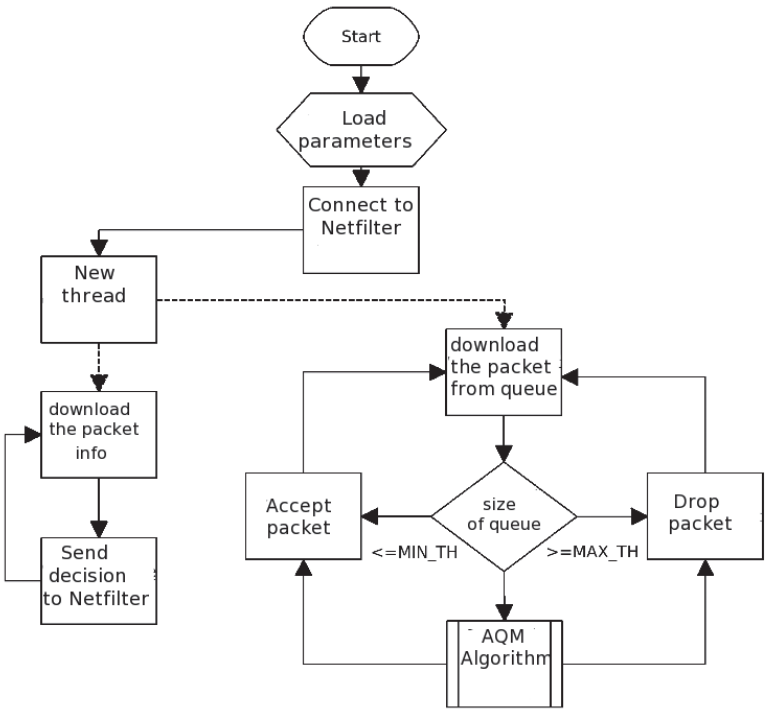

Fig. 11. Diagram of the program

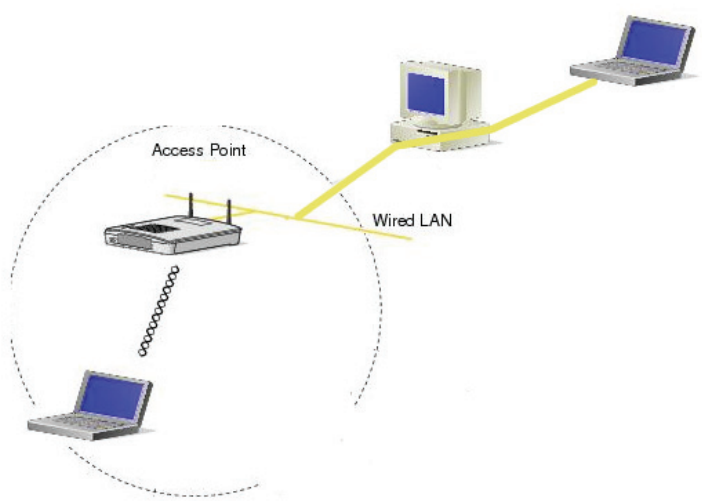

Fig. 12. The network used during the research

Figure 13. shows the buffer occupancy for CHOKe and gCHOKe algorithms. The algorithm gCHOKe fills up the queue below the parameter max_{th} = 15 and causes much greater fluctuations.

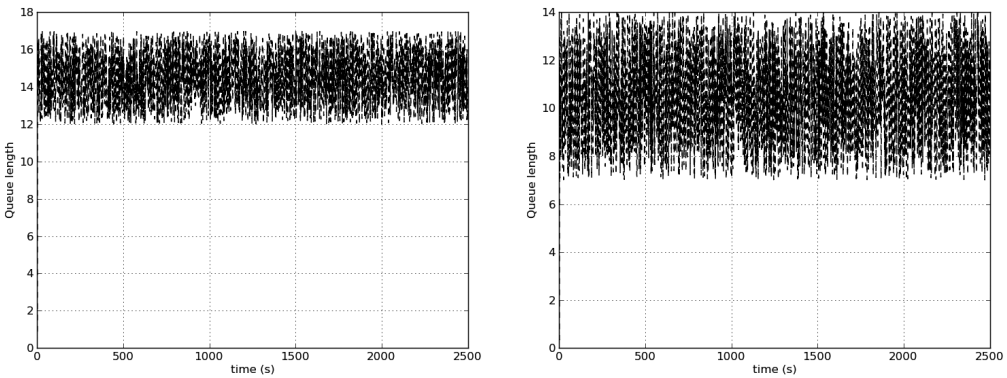

Fig. 13. The buffer occupancy for CHOKe (left) and gCHOKe (right) algorithms

5. Conclusions

In this paper we presented the problem of the CHOKe queue. We consider the problem of choosing the variant of the CHOKe algorithm for router queue behaviour. During the tests we analyzed the following parameters of the transmission with AQM: the number of rejected packets (by RED or CHOKe algorithm) and waiting times in queues. Our researches were carried out using the Discrette Event Simulator OMNET++. In the studies we also reconsider the problem of agressive (need more bandwith) sources. The simulations showed that the basic CHOKe and Front Choke algorithms rejected relatively small numbers of packets, while others practically throwing most of them. In the simulations, we took into account the small number of sources. For a small number of streams probability of selecting a good CHOKe victim is too large. Decrease in the number of losses with the number of sources can draw a conclusion that the CHOKe algorithms help to maintain the queue stability in the case of aggressive flows whose packets occupy the most space in the queue. In our study we have not considered the problem of sources with different intensity. We will deal with this issue in future researches. In this article we have additionally presented the behavior of this mechanisms involving the real working routers. Results obtained in the real network operation confirmed the results obtained in the simulation environment.

Acknowledgements

This research was partially financed by Polish Ministry of Science and Higher Education project no. N N516479640

References

1. D.R. Augustyn, A. Domański, J. Domańska, *Active Queue Management with non linear packet dropping function*, 6th International Conference on Performance Modelling and Evaluation of Heterogeneous Networks HET-NETs 2010.
2. D.R. Augustyn, A. Domański, J. Domańska, *A Choice of Optimal Packet Dropping Function for Active Queue Management*, Communications in Computer and Information Science, vol. 79, Springer 2010.
3. Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L.,Ramakrishnan, K., Shenker, S., Wroclawski, J., Zhang, L., *Recommendations on queue management and congestion avoidance in the internet*, RFC 2309, IETF (1998)
4. W. Chang Feng, D. Kandlur, and D. Saha, *Adaptive packet marking for maintaining end to end throughput in a differentiated service internet*, IEEE/ACM Transactions on Networking, vol. 7, no. 5, 1999.
5. J. Chen, F. Paganini, R. Wang, M.Y. Sanadidi, M. Gerla, *Fluid-flow Analysis of TCP Westwood with RED*, GLOBECOM 2004.
6. T. Czachorski, K. Grochla, F. Pekergin, *Stability and Dynamics of TCP-NCR (DCR) protocol in presence of UDP Flows*, in: Wireless Systems and Mobility in Next Generation Internet, LNCS no. 4396, pp.241-254, Springer 2007.
7. J. Domańska, A. Domański, T. Czachórski, *The Drop-From-Front Strategy in AQM'*, Lecture Notes in Computer Science, vol. 4712/2007, Springer Berlin/Heidelberg, 2007.
8. J. Domańska, A. Domański, T. Czachórski, *Implementation of modified AQM mechanisms in IP routers'*, Journal of Communications Software and Systems, vol. 4, no. 1, March 2008.
9. J. Domańsk, *Procesy Markowa w modelowaniu natężenia ruchu w sieciach komputerowych*, PhD thesis, IITiS PAN, Gliwice, 2005.
10. A. Eshete, Y. Jiang, *Generalizing the CHOKe flow protection*, Computer Network Journal, 2012.
11. C. V. Hollot, Vishal Misra, Don Towsley, *A control theoretic analysis of RED*, IEEE/INFOCOM, 2001.

12. C. V. Hollot, V. Misra, D. Towsley, W.-B. Gong, *On Designing Improved Controllers for AQM Routers Supporting TCP Flows*, IEEE INFOCOM 2002.

13. C. Kiddle, R. Simmonds, C. Williamson, B. Unger, H*ybrid packet/fluid flow network simulation*, Parallel and Distributed Simulation, 2003.

14. C. Liu, R. Jain, *Improving explicit congestion notification with the mark-front strategy*. Computer Networks, 35(2-3), 2000.

15. M. May, C. Diot, B. Lyles, J. Bolot, *Influence of active queue management parameters on aggregate traffic performance*, Technical report, Research Report, Institut de Recherche en Informatique et en Automatique, 2000.

16. V. Misra, W.-B. Gong, D. Towsley, *Fluid-based Analysis of a Network of AQM Routers Supporting TCP Flows with an Application to RED*, ACM SIGCOMM, 2000.

17. R. Pan, B. Prabhakar, K. Psounis, *CHOKe, A stateless AQM scheme for approximating fair bandwidth allocation*, IEEE INFOCOM, 942-952, 2000.

18. Ch. Panninder, Ch. Shobhit, G. Anurag, J. Ajita, K. Abhishek, S. Huzur, S. Rajeev, *XCHOKe: malicious source control for congestion avoidance at Internet gateways*, ICNP 2002.

19. Pengxuan Mao, Yang Xiao, Shaohai Hu, Kiseon Kim, *Stable parameter settings for PI router mixing TCP and UDP traffic*, IEEE 10th International Conference on Signal Processing (ICSP), 2010.

20. www.scipy.org.

21. S. Rahme, Y. Labit, F. Gouaisbaut, *An unknown input sliding observer for anomaly detection in TCP/IP networks*, Ultra Modern Telecommunications & Workshops, 2009.

22. L. Wang, Z. Li, Y.-P. Chen, K. Xue, *Fluid-based stability analysis of mixed TCP and UDP traffic under RED*, 10th IEEE International Conference on Engineering of Complex Computer Systems, 2005.

23. T. K. Yung, J. Martin, M. Takai, R. Bagrodia, *Integration of fluid-based analytical model with Packet-Level Simulation for Analysis of Computer Networks*, SPIE, 2001.

Analiza różnych wariantów mechanizmu CHOKe

Streszczenie

W artykule został przedstawiony model sieciowej transmisji danych poprzez router z zaimplementowanymi mechanizmami Aktywnego Zarządzania Kolejką (AQM). Badania zachowania mechanizmów AQM zostały przeprowadzone przy użyciu symulatora zdarzeń dyskretnych OMNET++. Uzyskane wyniki zostały zweryfiko-

wane w środowisku rzeczywistym. W oparciu o system operacyjny Linux stworzono programowy router implementujący mechanizmy wcześniej przebadane w środowisku symulacyjnym. Przeprowadzono analizę zachowania algorytmów AQM z rodziny CHOKe (CHOKe, xCHOKe, gCHOKe). W badaniach rozważano problem wpływu tych mechanizmów na tzw. agresywne (potrzebujące większego pasma) źródła transmisji danych.