ELECTRONICS AND ELECTRICAL ENGINEERING
ISSN 1392 – 1215 *2010. No. 7(103)*
ELEKTRONIKA IR ELEKTROTECHNIKA

TELECOMMUNICATIONS ENGINEERING
T 180
TELEKOMUNIKACIJŲ INŽINERIJA

Evaluation of TCP Acknowledgment Mechanism Influence on Router Performance

L. Pavilanskas, A. Statkus
Telecommunications Engineering Department, Vilnius Gediminas Technical University,
Naugarduko str. 41, LT-03227 Vilnius, Lithuania, e-mails: lukas.pavilanskas@el.vgtu.lt, arunas.statkus@teo.lt

Introduction

Nowadays with growth of data traffic [1], it is essential to control channel congestion efficiently in Internet for successful utilizing network resources (routers, switches, links, and etc). The main protocol for current objective is Transport Control Protocol (TCP) today. It is reliable connection-oriented protocol, which implements the flow control on heterogeneous network by means of a sliding window, congestion avoidance, and acknowledgment mechanisms, simultaneously. TCP runs over more network technologies than any other protocol and provides the highest degree of interoperability.

The main problem with TCP is that typical protocol implementations (RFC 3782) are successful at low speed (up to 100 Mbps) but it is unfit for high speed networks due to slow grow-up of congestion window, and poor evaluation of upper bound of the channel [2]. This is because the congestion window growth function – $W(t)$ has square features on initial phase (RFC 2581) and depends on channel delay: if latency is high, the congestion window growth rate will be low. To eliminate the following drawback, many TCP congestion control algorithms were developed. Most common, oriented for large volumes of data transfers, are the following: HSTCP (RFC 3649), Fast (RFC 3782), STCP (RFC 3286), Cubic [3].

All of these versions also rely on end-to-end ACK and congestion window [4], but general difference is that $W(t)$ is cubic-root on initial phase and depends on latency marginally, comparing with typical versions. The congestion window growth is defined in real-time and depends on last congestion event – received ACK [3]. In other words, the sender is allowed to increase the TCP data rate for each incoming ACK. As a result, unhampered transmission of ACK for successful flow control is very important, and any ACK rate suspension can influence the TCP functionality and performance [5–7].

In consequence, the growth of network bandwidth is coherent with ACK message rate on the following TCP operational scheme. When the network capacity is increasing, the ACK rate on the channel is increasing too. So, in high speed network we have another undesirable feature – growth of technological expenditures [8].

Moreover, TCP is mainly used in Internet for bulk traffic (HTTP, FTP, SMTP, IMAP, and etc) [9]. Bulk traffic sources are sending large packets in one direction only [10]. Therefore, uplink traffic is composed of short packets as showed in [5]. In such conditions, encapsulated Ethernet frame with TCP message without data (only ACK flag and number) is 64 Bytes, not including the preamble.

The poor performance of short frame traffic is well known and analyzed in detail [11], and [12]. This degradation occurs for two reasons: technological expenditures [8] and routers CPU overload [13]. The last concerns to the fact that router CPU load depend on the packet rate and not to the packet length [14]. It's clear, that on high speed networks this is observable better [13]. Therefore, with growth of bandwidth the TCP ACK message rate is increasing too. For a large number of short packets in traffic more efficient routers will be needed. Hereby, in following situation the routers with slow CPUs will become bottlenecks and will lead to the degradation of network performance too. The critical threshold of router functionality is ~ 45-50% of CPU load [15].

One of the ways to increase performance in modern high speed networks is by lowering the number of TCP ACK messages. This can be done through ACK filtering mechanism [16], which is well known and used in asymmetric and wireless networks environments [5], [6], [7], but is considered to be inconsistent with TCP traffic.

The aim of current paper is to find, how ACK filtering can influence the TCP functionality, and how significant number of TCP ACK messages affects the performance of the network channel – chain of the routers.

TCP acknowledgment mechanism

The acknowledgement mechanism is at the heart of TCP and relies on demand for the receiver to communicate with the sender by sending back an ACK as it receives data. TCP uses a cumulative ACK technique (RFC 813). The ACK number field in a TCP message header received by the sender indicates that all bytes of data with sequence

numbers less than that value have been successfully received (ACK'ed) by the destination application (Fig.1. segments *data1*, *data2* and *ack1*, *ack2*).

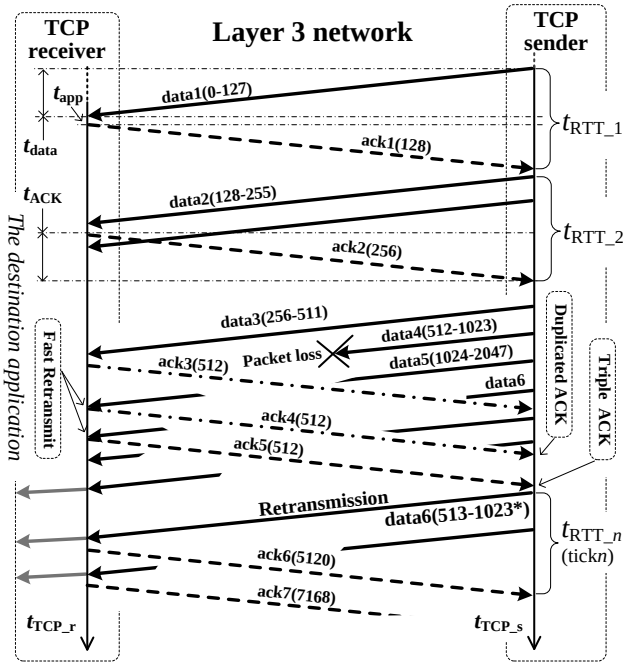

Fig. 1. TCP operation and acknowledgment mechanism diagram

Normally TCP does not send an ACK the instant it receives data. It delays ACK (RFC 2581), hoping to have data going in the same direction. A typical TCP uses delay of 200 ms and sends ACK for every other data message. However, there is not much TCP traffic with bidirectional data flows in Internet [9]. Delayed ACK is usable for an interactive applications (Telnet, SSH, and etc.) only [10].

The TCP sender's uses ACK to estimate how much data can be outstanding in the network without packets being lost in routers queues. For this purpose it uses a congestion window, which growth function *W(t)* depends on the feedback sender gets from the network through the received ACK. After initializing the TCP sender is allowed to increase the *W(t)* for each incoming ACK .

In this paper for research we use TCP Cubic were the congestion window is determined by function [3]

$$W(t)_{\mathrm{C}} = C\left(t - \sqrt[3]{\frac{W(t)_{\max}\cdot\beta}{C}}\right)^3 + W(t)_{\max}, \qquad (1)$$

where C is a scaling factor, t [s] – the elapsed time from the last window reduction, $W(t)_{\max}$ [bytes] – the maximal window size before the last reduction; β – a constant multiplication decrease factor applied for $W(t)_{\mathrm{C}}$ reduction to minimal at the time of loss event ($\beta \cdot W(t)_{\max}$) [3]

$$W(t)_{\min} = W(t)_{\max} - \beta \cdot W(t)_{\max}. \qquad (2)$$

Consequently, the ACK reception for TCP sender is very important process. Especially at initial phase, where growth of $W(t)_{\mathrm{C}}$ is very intense [3] and ACK loss can lead to degradation of the congestion window to its minimal value. Meanwhile, on phase when $W(t)_{\mathrm{C}}$ is close or equal to maximal value - $W(t)_{\max}$, growth becomes minimal (2) and congestion event becomes less important. In that case, ACK is mainly used for detection of data message loss and this is a second very important function of the TCP acknowledgment mechanism (RFC 813).

The TCP data message retransmission can occur because of two main reasons [17]: after expiration of retransmission timer (RFC 813) and after receiving three or more duplicated ACK (Fig. 1: *ack3*, *ack4*, *ack5*). In first case the retransmission timer expires when no new data is acknowledged for a set of threshold time (RFC 2581).

The retransmission timeout (t_{RTO}) is taken as a loss indication, and it triggers retransmission of the unacknowledged segments. The threshold time during which the confirming ACK message must be received is

$$t_{\mathrm{RTO}} = \overline{t_{\mathrm{RTT}}} + 4\cdot\sigma_{\mathrm{RTT}}, \qquad (3)$$

where $\overline{t_{\mathrm{RTT}}}$ is the average of time of successful TCP message transmission (Fig. 1: t_{RTT} measurements); σ_{RTT} – the root-mean-square of deviation of $\overline{t_{\mathrm{RTT}}}$.

If during t_{RTO} the ACK is not received, the data segments loss will be detected and the TCP sender will set $W(t)_{\mathrm{C}}$ to 1 segment (RFC 2988); since t_{RTO} indicate that channel utilization has changed dramatically [17].

Second reason occurs after receiving three or more duplicated ACK [17]. As shown in Fig. 1 the message *data4* was lost. The TCP receiver accepts messages *data3*, *data5*, *data6*, and so on (Fig.1.), but not *data4*. The *ack4*, *ack5* with same ACK number 512 were sent for received messages. After duplicated ACK is detected, the transmitter waits for t_{RTO} to expire. TCP does not yet know whether a duplicate ACK is caused by a loss or just reordering of segments. TCP sender waits the timeout (3) and assumes that if there is just a reordering it will not get ACK with number 512 anymore. However, after a while, a third duplicated ACK is received in a row (*ack5*). It is a strong indication that segments have been lost. TCP performs retransmission of segments 513-1023 with *data6*, without waiting for a t_{RTO} to expire. After this the sender maintains the number of outstanding segments by sending a new segment for each incoming ACK (1). It should be noted that only retransmission of identified as lost (timed-out) TCP segments, are implemented in the "conservative" TCP versions (RFC 3517). Meanwhile, in "aggressive" TCP implementation after the loss, all unacknowledged messages are retransmitted [17].

The TCP contains only a general assertion that data should be acknowledged promptly, but gives no more specific indication as to how quickly and as how frequently an ACK must be sent. In RFC's clearly is indicated, that current mechanism must maintain two very important functions: to prevent data retransmission, and as soon as possible to make ACK to permitting further data to be sent. In addition to this argument, the fact that ACK message are very important in the initial phase (1) of data transmission and the fact that rate of ACK rely on data rate (segments loss due to receiving 3 or more duplicated ACK) and cannot be less than $1/t_{\mathrm{RTO}}$ (3), must be evaluated.

ACK filtering technique

The concept of ACK filtering was discussed well in [16]. The idea is fairly simple. When router needs to send the current ACK, it scans queue for any early TCP ACK. If

ACK already exist in the queue, simply drop them before a queuing the new ACK. Since acknowledgments are cumulative, the newest one obsoletes all older ACK. So, there's no need for more than one per connection to ever be in the queue. Dropping old ACK when a new one is queued means there would never be more than one ACK on the queue at any time, so this is pretty much the same as replacing the earlier ACK with the newer one.

In the same reference [16] the main drawbacks of ACK filtering were reviewed. It is noted that ACK clocking scheme, which for both drop and congestion control is used, with the ACK filtering can be destroyed. The current propositions are compelling. However, the discussed apprehensions can be challenged with arguments of [5], where the TCP performance in the asymmetric links was analyzed. It is shown that asymmetry affects the TCP performance, because it relies on feedback of cumulative ACK from the receiver. In addition, typical TCP is ACK clocked, so the arrivals of ACK on the reverse channel have significant effects on the forward channel throughput. In the networks with bandwidth asymmetry the ACK filtering can work well. This improves the forward TCP throughput and the fairness of competing connections greatly. In paper ACK filtering was modeled on Opnet.

Work [6] presents a quite similar study. Analysis of unfairness problem between TCP upstream data and ACK downstream on the unevenly shared wireless channel is provided. It is shown that ACK filtering increases 802.11 channel utilization without any dependence on t_{RTO} (3).

The extensive simulations with ACK filtering in [7] were proposed. It is demonstrated that lowering ACK number can improved TCP performance significantly: achieving up to 25% gain in chain networks and 35% in a complex grid network, compare with typical TCP. In work the ACK filtering motivation follows from fact that short ACK messages consume channel capacity comparable to data packets when the transmission is high rate.

The methodology of experiments

In order to find out, what real influence the ACK filtering makes to the TCP functionality and how it affects the performance of the network channel devices (routers) an experiments were performed. For this reason Ethernet network with IP routing, and TCP session between two independent nodes (PC0 and PC1/PC2, Linux OS, 2.6.32 kernel, and TCP Cubic version enabled [3]) were created. The structure of network is presented in Fig. 2.

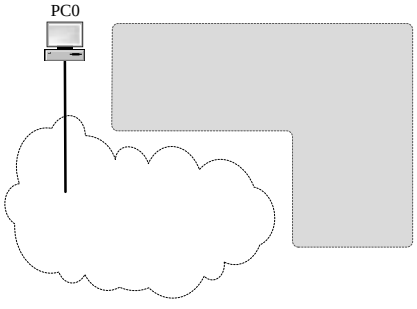

Fig. 2. Structure of network used in experiments

In current network the following equipment has been used: PC0 as FTP server – the transmitter of data TCP messages, and PC1/PC2 as FTP client – receiver of TCP data messages (TCP ACK sender), the transparent Ethernet bridge – BR, target router – R, and additional router – R0 (Cisco 881) as well. For experiments two routers of different generations Cisco 881 and Cisco 1841 were used. The main difference between them is CPU power. All devices were connected with 100 Mbps Ethernet links (100BaseTx). The ACK filtering was implemented on BR device. It has been created on Linux based (2.6.32 kernel) PC with Ethernet bridging <*brctl*> application. This tool was taken because Linux bridging is faster, work as simple switch, and don't make significant impact to flow parameters comparing with routing.

The ACK filtering has been pursued only in one direction from PC1/PC2 to PC0, whereas in opposite direction the traffic passed through BR without any alterations. Filtering was based on exact frame rate control with Committed Information Rate (CIR) and Committed Burst Size (CBS=5 kB). The last one was used to avoid degradation of congestion window on initial growth phase (1). Filtering was made using <*tc*> application, which drop/policed ACK messages if it exceeds specified rate. The scripting code of ACK filtering is shown in Fig. 3.

```
01 tc qdisc add dev eth1 ingress
02 tc filter add dev eth1 parent ffff:0
       protocol all prio 1 u32 match u32
       0xaff0001 0xffffffff at 16 classid
       ffff:0 police index 2 rate 12500bps
       burst 102400 mpu 0 action drop/pass
03 tc filter add dev eth1 parent ffff:0
       protocol all prio 1 u32 match u32
       0x0 0x0 at 0 classid ffff:0 police
       index 3 rate 1bps burst 1 action
       drop/drop
```

Fig. 3. Scripting source of *tc* policing

The second router (R0) has been used on purpose to keep more realistic IP based network with all routing and switching functionalities. The target router parameters during the experiments with SNMP protocol were collected throw independent router interface. For data transmissions the FTP application has been used. In all experimental iterations file of 400 MB size was transferred. The data speed was controlled on PC0 (FTP server) with <*tc*> script, which shaped to desirable speed without packet loss (delay of traffic only). For this Token Bucket Filtering – TBF (RFC 5624) was used.

Performance evaluation experiments

The target of first experiment scenario was to find how the ACK filtering can influence the TCP performance and data transfer integrity. For this, a file of 400 MB size from PC0 to PC1 was repeatedly transferred (Fig. 2.). On the each iteration the ACK filtering rate (0%, 20%, 40%, 60%, and 80%; values of the maximal ACK rate without filtering) was changed.

As shown in Fig. 4 TCP message rate for entire period was stable in all iterations. The growth of message rate (1) is high and equal at all ACK drop values (events

up to ~3s). This occurred because of two reasons: $W(t)_C$ of used TCP version slightly depends t_{RTO} (3) and ACK filtering is activated only after CBS is exceeded – when the $W(t)_C=W(t)_{max}$ (1). At the end of transfer we have "decline" – the finish of data transfer.

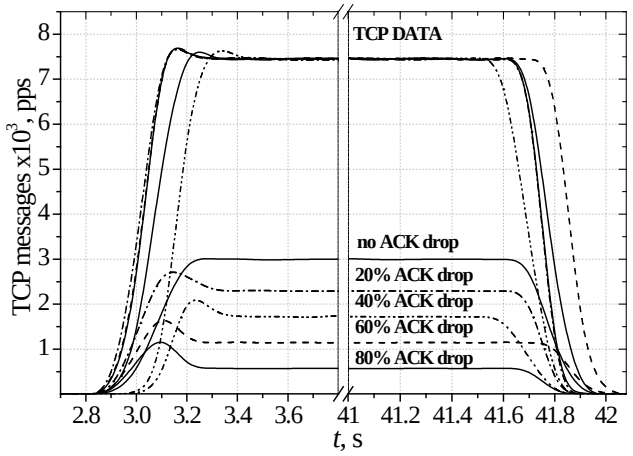

Fig. 4. TCP messages rate during file transmission for various ACK drop values

The result shows that ACK filtering does not affect data transfers of single TCP session. We observed that transfer remains stable for up to 80% of ACK drops. However, if losses are above 85% the ACK rate becomes less than $1/t_{RTO}$ (3), and data rate degrades fatally.

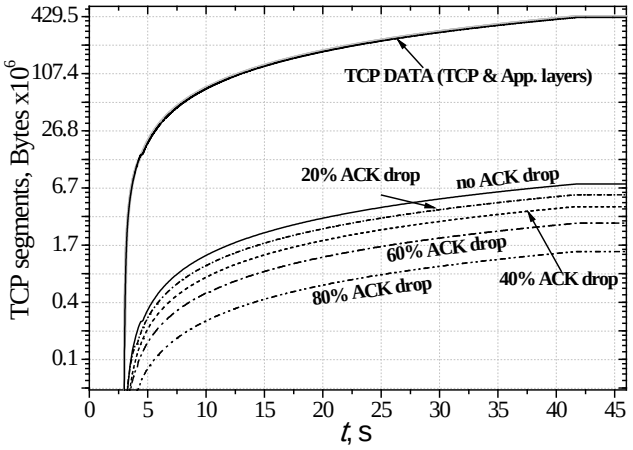

Fig. 5. Cumulative growth of TCP segments count during file transmission for various ACK drop values

The comparison of cumulative TCP segments (count of segments on TCP layer) and FTP bytes (count of data on FTP layer) rates during period (t) of file transfer is presented in Fig. 5. It is shown that on FTP and TCP layers (lines are coincident) the same amount of bytes was received at any given time period. Consequently, it is possible to do the suggestion, that count of duplicated ACK and TCP data retransmissions (3) are not increased respectively (Fig. 1: $t_{RTT}=t_{data}+t_{ACK}$ is less then t_{RTO}).

The second experiment goal was to find what kind of influence cumulative ACK mechanism has on network equipment performance. For this purpose a file transfer of 400 MB, was performed and CPU load of router (Cisco 881, Cisco 1841) was measured (Fig. 2.). The experiments were performed at various data rates (1, 4, 8, 16, 32, 65, and 90 Mbps) in scenarios with and without filtering and in scenario when 80% of ACK is filtered.

As shown in Fig. 6 the CPU load is decreasing when the ACK filtering is used. The same situation is observable for both routers. It is clear that results do not depend on router type and amount of data in TCP message. It depends on amount of processed packets by the router CPU. This is confirming the results presented in [14]. Moreover, in Fig. 6 is observable that CPU load utilization is linear, and depends on frame rate: if the TCP data rate increases, the ACK rate is increasing too. The functions of the CPU load curves are quite similar for both routers. The difference is only in designed CPU's power.

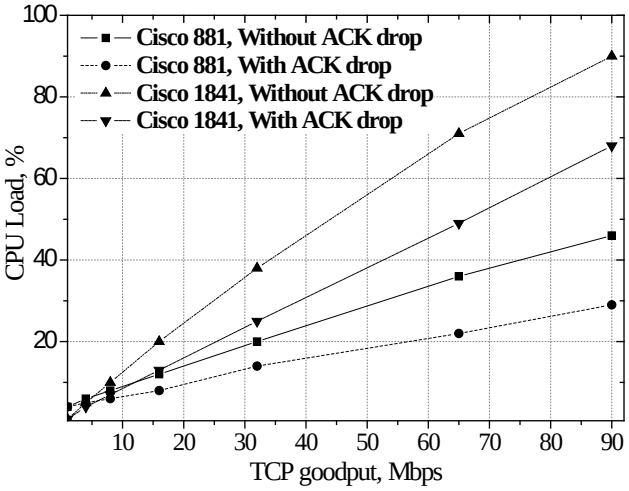

Fig. 6. Target routers CPU load for various FTP traffic rate (TCP goodput) when 80 % ACK is dropped

The relation between target router performance and CPU load is shown in Fig. 7. Performance increase should be understood as relative CPU load reduction caused by employing of ACK filtering. With ACK drop of 80% the performance can be increased by 30%, comparing with case when CPU load is 25%, and ACK filtering is not used. Meanwhile, if we have the CPU load of 60% (more than routers overload threshold [14]) with 80% of ACK drop we can increase the performance by 32%. It means that in current situation CPU load will be approx. 40%.

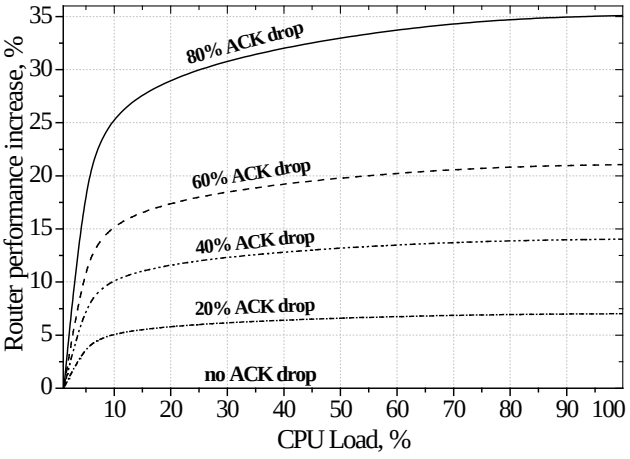

Fig. 7. Performance increase on target router CPU load for various ACK drop values

The third experiment issue was to find how the ACK filtering can influence two concurrent and independent TCP sessions. During investigation the PC0 was used as FTP server and PC1/PC2 - as FTP clients. With each TCP session the 400 MB files were transferred. The ACK filtering was performed on both sessions simultaneously.

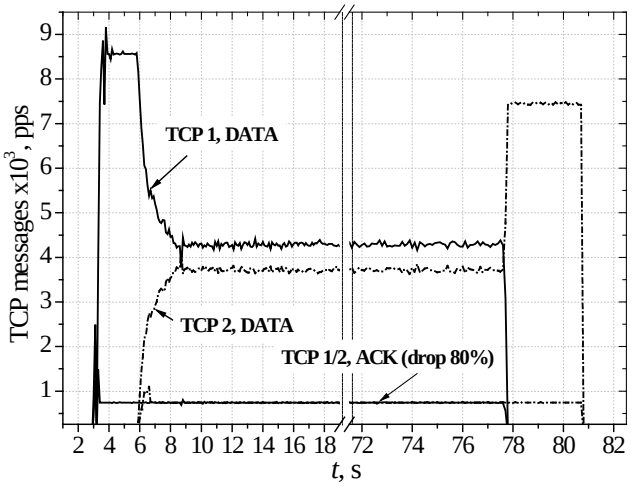

Fig. 8. TCP message rate during two files transmission: both sessions with 80% of ACK drops

The dependence of two TCP sessions message rate during file transmissions is presented In Fig. 8. Graph clearly shows that both independent sessions remain concurrent and divide channel almost equally. In fact, this cannot be so, because TCP objective is not equal channel sharing. At 3s, as in situation with one session (Fig. 4), the TCP1 begins to increase the message rate according to $W(t)_C$ (1). While at 6s TCP2 session is starting too, and about 9s the message rate of both sessions becomes approx.: $R_{TCP1} \approx 44$Mbps; $R_{TCP2} \approx 49$Mbps; $R_{TCP} \approx 93$Mbps. While the TCP1 session is degraded due to congestion (2).

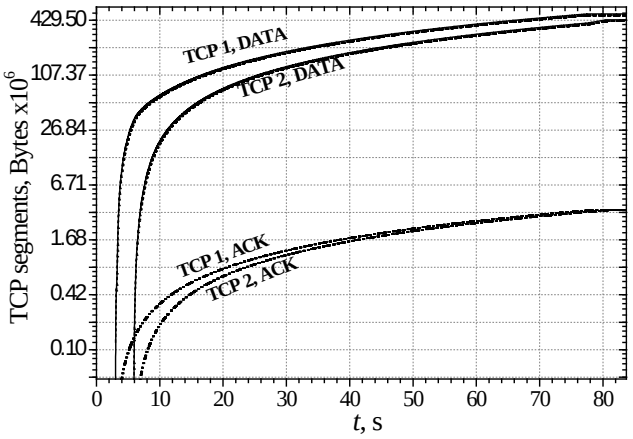

Fig. 9. Cumulative growth of TCP segments count during two file transmission: both sessions with 80% of ACK drops

After 78s the TCP1 rate is decreasing since finishing of file transfer, while the TCP2 conversely starts to grow-up rate (1). At 81s TCP2 is finishing transfer too. Current fine competition between two independent session's show, that both of them from TCP point of view are working well, and the ACK filtering does not make significant influence on TCP functionality in current conditions.

The dependence of cumulative data during two file transmissions is shown in Fig. 9. It can be seen that both sessions collect messages well in TCP layer and in application layer (Fig.1.). It means that TCP goodput (transmitted data to upper layer) of both sessions are close to maximal, while the count of duplicated ACK is minimal.

TCP interoperability defines whether a protocol is fair to other TCP sessions. Therefore, it's important to find how ACK filtering increases unfairness of TCP. For this purpose fourth experiment was performed. The scenario was the same as in previous, only the filtering for one session and the PC0 without shaping was used (Fig. 2.).

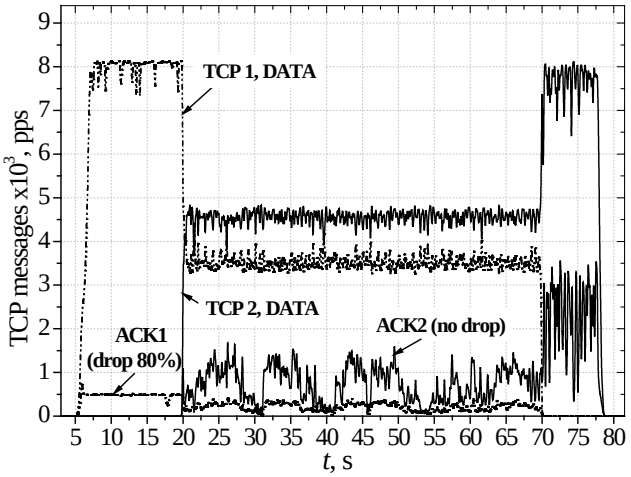

Fig. 10. TCP message rate during two files transmission: one session without ACK drop, and other with 80% of drops

The dependence of message rate of two TCP sessions on file transmissions time is presented in Fig. 10. Graph shows that both sessions divide channel almost equally as in previous scenario. The insignificant unfairness between sessions was observed. But this is typical case for real network. To understand unfairness better, the extensive analysis must be done with various TCP versions. And this is the main issue of future investigations.

Conclusions

In current work the investigation of TCP functions and ACK filtering was presented. It is shown that TCP congestion window and acknowledgment mechanism are dependent processes with common variables. Therefore, the growth of network bandwidth is coherent with ACK rate: when the network capacity is increasing, the ACK rate on channel is increasing too. This leads to network equipment CPU performance degradation. To resolve this weakness the usage of ACK filtering was proposed.

The experiments were performed to find how significantly number of ACK affects the router CPU and how impact of ACK filtering can influence the TCP functionality and performance. We conclude that:

1. ACK filtering does not significantly affect the TCP data transfer in normal network conditions. The TCP data transfer remains stable for up to 80% of ACK drops. However, if losses are above 85% the ACK rate becomes less than $1/t_{RTO}$, and session is terminating immediately.

2. The performance of router CPU depends on ACK count and can be increased with ACK filtering. The CPU

load utilization is linear, and depends on data rate: if the TCP data rate increases, the ACK rate is increasing too.

3. On CPU load of 25% with 80% of ACK drop, it is possible to increase the router performance by 30%. Meanwhile, on CPU load of 60%, on the same filtering conditions, the performance can be increased by 32%.

4. The ACK filtering can be used not only with single TCP session but also with concurrent sessions. Results show that two sessions work well, without any evident signs of the instabilities; although a slight unfairness among TCP sessions were observed.

In conclusion, the presented results show that ACK filtering does not affect the functionality of TCP in normal network conditions, but allows more efficient use of network equipment. Therefore, to implement the suggested solution the investigation of the ACK filtering influence on flows of other popular TCP versions, and the analysis of usage filtering on lossy channels, must be performed. These issues are the main of our future investigations.

References

1. **Eidukas D., Valinevičius A., Vilutis G., Kilius Š., Vasylius T.** Information Network Loading Evaluation // Electronics and Electrical Engineering. – Kaunas: Technologija, 2005. – No. 8(64). – P. 22–26.
2. **Dalton L. A., Isen C.** A Study on High Speed TCP Protocols // Proc. IEEE: GLOBECOM. – 2004, – Vol. 2. – P. 851–855.
3. **Sangtae H., Injong R., Lisong X.** CUBIC: A New TCP–Friendly High–Speed TCP Variant // Proc. IEEE: SIGOPS. – 2008. – Vol. 42. – No. 5. – P. 64–74.
4. **Jacobson V.** Congestion avoidance and control // in Proc. Conference IEEE: SIGCOMM. – 1988. – Vol. 1. – P. 314–329.
5. **Wu' H., Wu' J., Cheng' Sh., Ma J.** ACK Filtering on Bandwidth Asymmetry Networks // Proc. IEEE: APCC/OECC. – 1999. – P. 175–178.
6. **Keceli F., Inan I., Ayanoglu E.** TCP ACK Congestion Control and Filtering for Fairness Provision in the Uplink of IEEE 802.11 Infrastructure Basic Service Set // in Proc. Conference IEEE: ICC. – 2007. – P. 4512–4517.
7. **Chen B., Marsic I., Shao H–R., Miller R.** Improved Delayed ACK for TCP over Multi–hop Wireless Networks // Proc. IEEE: WCNC. – 2009. – P. 1–5.
8. **Kajackas A., Pavilanskas L.** Analysis of the Technological Expenditures of Common WLAN Models // Electronics and Electrical Engineering. Kaunas: Technologija, 2006. – No. 8(72). – P. 19–24.
9. **Kim, H., Claffy, K., Fomenkov, M., Barman, D., Faloutsos, M., Lee, K.** Internet traffic classification demystified: myths, caveats, and the best practices // In Proc. IEEE: CoNEXT. – New York: ACM, 2008. – P. 1–12.
10. **Caceres R., Danzig P. B., Jamin S., Mitzel D. J.** Characteristics Of Wide–Area TCP/IP Conversations // Proc. IEEE: ACM SIGCOMM. – 1991. – P. 101–112.
11. **Xiao Y., Rosdahl J.** Throughput and delay limits of IEEE 802.11 // IEEE: Comm. Letters, 2002. – Vol.6. – P. 355–357.
12. **Rindzevicius R., Tervydis P., Narbutaite L., Pilkauskas V.** Performance Analysis of Data Packet Transmission Network with the Unreliable Transmission Channels // Electronics and Electrical Engineering. – Kaunas: Technologija, 2008. – No. 4(84). – P. 53–58.
13. **Bencivenni M., Carbone A., Fella A., Galli D., Marconi U., Peco G., Perazzini S., Vagnoni V.** High rate packet transmission on 10 Gbit/s Ethernet LAN using commodity hardware // Proc. NPSS: Real Time. – 2009. – P. 167–182.
14. **Paredes–Farrera M., Fleury M., Ghanbari M.** Router Response to Traffic at a Bottleneck Link // in Proc. 2nd Int. Conference IEEE: TridentCOM. – 2006. – P. 4–41.
15. **Ogawa Y., Nakaya A.** Estimating the Performance of a Large Enterprise Network for the Updating of Routing Information // in Proc. Conference Workshop IEEE: IP Operations and Management. – 2002. – P. 161–165.
16. **Karn P.** Dropping TCP ACK's // Mailing list. – 1996 [online: ftp://ftp.isi.edu].
17. **Jacobson V.** Congestion avoidance and control // in Proc. Conference SIGCOMM' 88: ACM Special Interest Grp. on Data Comm. – New York: ACM, 1988. – P. 314–329.

Received 2010 02 11

L. Pavilanskas, A. Statkus. Evaluation of TCP Acknowledgment Mechanism Influence on Router Performance // Electronics and Electrical Engineering. – Kaunas: Technologija, 2010. – No. 7(103). – P. 95–100.

The efficient usage of congestion control becomes significant as Internet traffic grows. The quality of congestion control greatly affects utilization efficiency of the data network equipment. This paper presents detailed analysis of interaction between congestion control and acknowledgment mechanism functions. Also analysis of possibilities to reduce the number of ACK messages with ACK filtering is presented. The presented results of experiments with real network are shows that data network router performance, without any considerable influence to TCP functionality can be significantly increased when the ACK filtering is used. Ill. 10, bibl. 17 (in English, abstracts in English, Russian and Lithuanian).

Л. Павиланскас, А. Статкус. Оценка влияния механизма TCP подтверждения на производительность маршрутизатора // Электроника и электротехника. – Каунас: Технология, 2010. – № 7(103). – С. 95–100.

При увеличении интернет трафика увеличивается и значение управления перегрузками сети. От качества управления значительно зависит эффективность использования оборудования сетей передачи данных. В этой статье детально анализируются взаимодействия функций механизмов контроля переполнения и подтверждения, а также возможности уменьшения генерируемых сообщений ACK при помощи фильтрирования. Предоставленные эксперименты на реальной сети подтверждают, что используя фильтрирование ACK возможно значительно увеличить производительность маршрутизаторов сетей без значительного воздействия на функциональность TCP. Ил. 10, библ. 17 (на английском языке; рефераты на английском, русском и литовском яз.).

L. Pavilanskas, A. Statkus. TCP patvirtinimo mechanizmo įtakos maršrutizatoriaus našumui nustatytmas // Elektronika ir elektrotechnika. – Kaunas: Technologija, 2010. – Nr. 7(103). – P. 95–100.

Didėjant interneto srautams, didėja ir tinklo perkrovų valdymo reikšmė. Nuo valdymo kokybės labai priklauso duomenų perdavimo tinklų įrenginių naudojimo efektyvumas. Šiame straipsnyje detaliai analizuojamos TCP protokolo perkrovimų valdymo ir patvirtinimo mechanizmų funkcijų sąveika bei galimybės filtravimu sumažinti generuojamų ACK pranešimų skaičių. Pateikti realiame tinkle atliktų eksperimentų rezultatai rodo, kad, filtruojant ACK ir nedarant didelės neigiamos įtakos TCP funkcionalumui, galima gerokai padidinti duomenų perdavimo tinklo maršrutizatorių našumą. Il. 10, bibl. 17 (anglų kalba; santraukos anglų, rusų ir lietuvių k.).

DOI: 10.5755/j02.eie.9287