WTFC to TCP Flow Control Proxy

Vesna Pekic, Ante Kristic, Julije Ozegovic
Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture
University of Split
Split, Croatia
e-mail: vesna.pekic@fesb.hr, ante.kristic@fesb.hr, julije.ozegovic@fesb.hr

Abstract—**Integration of different kinds of traffic on the Internet can be resolved by employing QoS policies on access networks. One possible solution is to use WTFC (Window-Time Flow Control) flow control algorithm on access network. In this paper, the WTFC-TCP flow control proxy is introduced in order to exploit the ability of WTFC to estimate the optimal working point for the available capacity and to keep on average empty node buffers. A basic proxy model is defined and elementary congestion signaling algorithms for flows directed to and from access network are applied. Simulation experiments have shown potential successfulness of the concept.**

Keywords-access network; WTFC; QoS; Flow Control Proxy

I. INTRODUCTION

WTFC (Window-Time Flow Control) [12] was designed to keep a network at the optimal working point, using its fair share of the available capacity and keeping node buffers empty. This makes it suitable for integrated services implementation. The problem is the coexistence of WTFC with the more aggressive TCP (Transport Control Protocol). On the other hand, TCP is the prevailing technology that will be hard to replace in the near future. One possible solution is to introduce new technologies on access network without changing TCP/IP protocol stack on the rest of the network.

The idea of this paper is that the cooperation of WTFC on access network and TCP on the rest of the network would be beneficial for deployment of integrated services. For that purpose, a special gateway or proxy node is needed to transfer packets from one protocol to the other, and to map their flow controls. We explore the possibility of interconnecting WTFC with TCP and show some simulation results using WTFC-TCP flow control gateway/proxy.

In Section 2, an overview of related work on improving TCP performance via a proxy and on mapping TCP to another protocol is given. A brief description of TCP and WTFC flow control is given in Section 3. Section 4 introduces a model used for WTFC-TCP interconnection, highlights the problems encountered when mapping the two protocols' flow control and proposes basic algorithms to successfully perform the mapping. Simulation results are given in Section 5 and conclusions with future work in Section 6.

II. RELATED WORK

Intensive work has been done on improving TCP performance over a combination of wireless access and fixed transport network. The dominant technology, if fixed side TCP client is not to be altered, is using PEPs (Performance Enhancing Proxies). A survey of existing performance enhancing proxy schemes is given in [4]. PEP schemes in the literature include modifying acknowledgment (ACK) spacing [3], generating local acknowledgments for large bandwidth-delay products (BDP) [14], variable bandwidth [7], or wireless links [9], specifically negative acknowledgments, and performing local retransmission [1][3][9], as well as ACK filtering and reconstruction [2].

The accent in the above schemes is on error recovery and hiding errors caused by unreliable nature of wireless links from TCP on the fixed side, so that standard TCP congestion control wouldn't unnecessarily trigger and reduce congestion window. Proposals concerning actual flow control are receiver window signaling according to free space in proxy buffers [1][14], ACK spacing [3] and changing window according to congestion measurements [13]. Aggregating flows that pass through the same Base Station to the same mobile host and using aggregate state variables instead of per-flow state variables is explored in [5].

Several approaches, like [1], suggest splitting connection between fixed host and mobile host at the Base Station. A protocol optimized for wireless links is employed on the wireless part of the network. Our work follows a similar idea of using different protocols for different parts of the network. However, even though applying WTFC-TCP connection on heterogeneous wired-cum-wireless networks looks promising, due to WTFC flow and error control being decoupled, in this paper a general type of link is considered. Interconnecting congestion controls of TCP and another protocol is described in [6]. An XCP-TCP gateway is used that interconnects XCP (eXplicit Control Protocol) and TCP networks. It maps the congestion control functions between TCP and XCP by mapping incipient congestion of XCP domain into drop event of TCP domain.

III. TCP AND WTFC FLOW CONTROL MECHANISMS

A. *TCP Flow Control*

Transport Control Protocol (TCP) uses window flow control consisting of four main mechanisms: slow start (SS), congestion avoidance (CA), fast retransmission and fast recovery, although many more have been proposed in the literature.

TCP sender maintains two variables that determine window size and the rate of its increment: congestion window and slow start threshold [8]. Congestion window (*cwnd*) represents the amount of data bytes that can be transmitted at a given time before receiving an acknowledgment (ACK) from the receiver. Slow start threshold (*ssthresh*) serves as a boundary that determines if slow start or congestion avoidance mechanism is used to control data transmission. *Ssthresh* can be recognized as the sender's estimation of optimal packet window.

Slow start mechanism is used while *cwnd<ssthresh*, i.e. at the beginning of a transfer, or after detecting network congestion. Packet loss or ECN (Explicit Congestion Notification) serve as congestion indication. Initially, TCP sender sets *ssthresh* to receiver's advertised window and *cwnd* to 1 packet and increases it by 1 for each ACK received, effectively doubling *cwnd* every round trip time (RTT). After packet loss, *ssthresh* is set to half the current *cwnd* value and slow start begins again from its initial *cwnd*.

When *cwnd* value exceeds *ssthresh*, TCP sender transits from slow start to congestion avoidance mechanism. During congestion avoidance, *cwnd* is incremented by approximately 1 segment per RTT. Congestion avoidance continues until a packet loss is detected.

Basic indication of packet loss is the occurrence of retransmission timeout (RTO). RTO is dynamically computed based on RTT value and its variance. Fast retransmission algorithm shortens the time needed to detect packet loss. After receiving an out-of-order packet, TCP receiver sends a duplicate ACK to inform the sender of a possible packet loss. When the sender receives three duplicated ACKs, it concludes that the packet has been lost. Sender can then perform a retransmission of what appears to be the missing packet, without waiting for RTO to expire.

To benefit from fast retransmission, fast recovery mechanism was introduced. After a packet loss detection and fast retransmission, values of *ssthresh* and *cwnd* are set to cwnd/2 and ssthresh+3, respectively. This takes into account packets that have left the network and have generated three duplicated ACKs. After receiving an ACK for the retransmitted packet, the sender sets *cwnd* to *ssthresh*.

B. *Window Time Flow Control*

Window Time Flow Control (WTFC) uses window and RTT measurements to calculate optimal window and packet sending rate [12]. *WT* (window-time) space is used for analysis. *W*,*T* plane is determined by the network optimal working point (W_0, T_0), T_0 and W_0 being minimal RTT and optimal window in case of only one user, respectively. This point represents the total capacity of the network from the packet transmitter's point of view. $\mu=W_0/T_0$ would then be the optimal sending rate.

When more users share the network, each should acquire a proportional part of the network capacity, $\mu_j = \alpha\mu$, under the assumption that $\alpha \in (0,1)$ is equal for all users. Optimal working point ($W_0(\alpha)$, $T_0(\alpha)$) of each transmitter, that now observes reduced capacity μ_j, depends on (W_0, T_0) and the parameter α. The pair ($W_0(\alpha)$, $T_0(\alpha)$) also determines delay for each transmitter:

$$T = \begin{cases} T_0(\alpha) & ,W \le W_0(\alpha) \\ W\dfrac{T_0(\alpha)}{W_0(\alpha)} & ,W > W_0(\alpha) \end{cases} \tag{1}$$

Parameter α defines a set of delay response curves in WT space. Their breaking points, being at the same time optimal working points, are placed along the hyperbola-like curve defined by constant bandwidth delay product; see Figure 1.

When packet acknowledgment arrives, transmitter measures new coordinates in *W*,*T* plane, which allows it to compute α. Then it calculates the optimal working point C, i.e. optimal sending period $t_0(\alpha)$ and window $W_0(\alpha)$, reducing the packet rate from the point A according to

$$t_0(\alpha) = \frac{T}{W} \quad , \quad W_0(\alpha) = T_p\frac{W}{T} + 1 \tag{2}$$

or increasing it from the point B according to:

$$t_0(\alpha) = T - T_p \quad , \quad W_0(\alpha) = \frac{T_p}{T - T_p} + 1 \cdot \tag{3}$$

T_p in the above expressions is the propagation delay. WTFC transmitter requires information about the total network capacity (W_0, T_0) for WTFC to function. Network nodes can signal parameters needed to compute the capacity of the bottleneck link using special fields in packet headers. Alternatively, transmitter can estimate the capacity [11]. Experimental simulations in [11] have shown that signaling yields better fairness. A special Fast-Start algorithm is used for sending packets before the first acknowledgment arrives and optimal working point can be computed.

WTFC does not use packet loss to detect congestion and simulation measurements have shown that WTFC algorithm can provide high network utilization with on average empty node buffers [12].

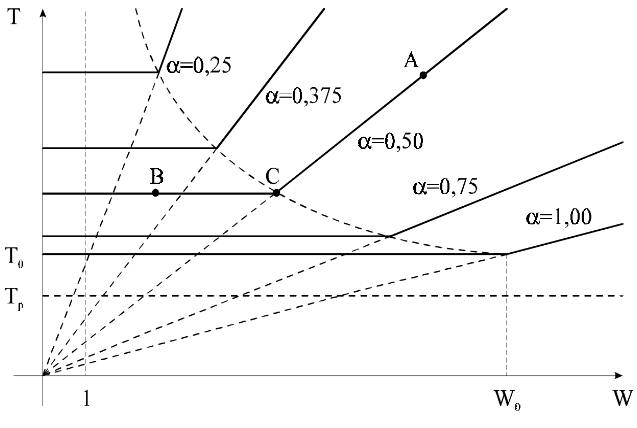

Figure 1. Delay curves depending on α in WT space (from [10])

IV. Connecting TCP With WTFC

The goal of our concept is to exploit the ability of WTFC to estimate the optimal working point for its share of the available capacity, while keeping the property of TCP to compete for its share of the total capacity.

A. Basic connecting architecture

Model of the access network connected to the Internet provider network and core Internet is used as the basic connecting architecture. The model is further developed using packet queues abstraction; see Figure 2. On both WTFC and TCP domains, WTFC and TCP sender packet queues are considered. Of the four, two are located on the proxy node. The WTFC and TCP flows are interconnected through the interconnection queues located on the proxy node. Interconnection queues, called proxy buffers, are available for manipulation.

The WTFC-TCP flow control proxy is a stateful (per flow) proxy and all the traffic of an access network is supposed to pass through it. Its placement could be in DSL Access Multiplexers or in Base Stations of cellular networks. Those are highly distributed devices placed relatively near the end-user, with moderate several hundred flows passing through at a given time.

We consider four main scenarios:

- TCP to WTFC domain flow; congestion on the TCP sender domain.
- TCP to WTFC domain flow; congestion on the WTFC receiver domain.
- WTFC to TCP domain flow; congestion on the WTFC sender domain.
- WTFC to TCP domain flow; congestion on the TCP receiver domain.

In case of congestion on the sender side, it is enough to simply translate the packet from one protocol to the other on the proxy and forward it immediately, since the proxy buffer will not be saturated. Intervention is needed when congestion is on the receiver side. In that case, packets will begin to accumulate in the proxy buffer and some sort of mechanism should slow the sender down, while at the same time keeping the efficiency high.

Since both WTFC and TCP protocols use acknowledgments (ACKs) to estimate network condition, ACK manipulation is the first choice to achieve desired performance. Three different possibilities are detected, to close acknowledgments loopback at the near-end (e.g. proxy TCP receiver to TCP sender), at the far-end (e.g. WTFC receiver to the TCP sender), and on the proxy buffer (e.g. proxy WTFC receiver to WTFC sender after a packet is removed from the WTFC-TCP proxy buffer).

Proxy buffer packet losses are acceptable with far-end ACK loopback, because those buffers are included in the normal queue chain of the packet trajectory. However, with near-end and buffer ACK policies, proxy buffer packet loss is unacceptable, because it is not covered with the packet loss recovery in either WTFC or TCP domain.

B. Basic algorithms for TCP to WTFC domain flows

In case of TCP to WTFC domain flow, TCP sender is located at the distant node and TCP receiver at the proxy node, while WTFC sender is located at the proxy node and WTFC receiver at the distant node. In case of congestion in TCP domain, TCP should react normally to packet loss. When congestion arises in WTFC domain, proxy WTFC sender should reduce its sending rate using normal WTFC mechanisms. However, TCP sender continues to use its current congestion window and TCP-WTFC proxy buffer tends to overflow.

When far-end ACK loopback is used, ACKs in TCP domain are sent after corresponding ACKs from WTFC domain are received at the proxy, so that TCP sender can have accurate overall RTT information.

Proxy WTFC sender can calculate optimal sending interval and window for WTFC domain using either capacity estimation or explicit signaling. TCP-WTFC proxy buffer is used between TCP and WTFC domain to forward packets with the optimal interval computed. ACKs are forwarded directly between WTFC and TCP domain. Their spacing paces TCP sender to optimal sending interval. Still, TCP sender increases its *cwnd*, either doubling it per RTT if in SS phase, or incrementing it by one per RTT if in CA phase. Consequently, packets start accumulating in the TCP-WTFC proxy buffer.

Despite the reason for TCP-WTFC proxy buffer overflow, it is obvious that some kind of WTFC to TCP congestion signaling is needed. TCP reacts to packet losses, so packets are intentionally discarded from the TCP-WTFC proxy buffer. The algorithm is triggered when TCP-WTFC proxy buffer reaches two times the proxy WTFC sender

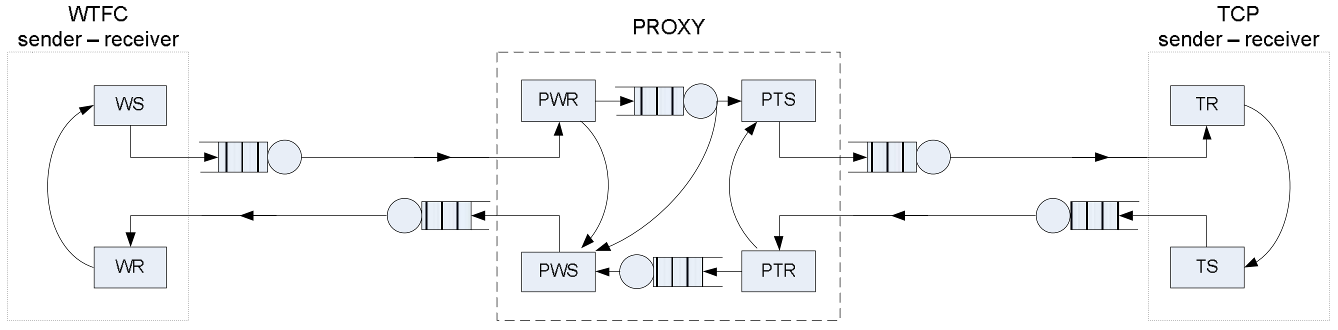

Figure 2. WTFC-TCP flow control proxy model

congestion window.

Simulation results show acceptable performance of the proposed far-end ACK policy with the induced packet losses algorithm, so near-end and buffered ACK policies are left for future research.

C. Basic algorithms for WTFC to TCP domain flows

In case of WTFC to TCP domain flow, WTFC sender is located at the distant node and WTFC receiver at the proxy node, while TCP sender is located at the proxy node and TCP receiver at the distant node. In case of congestion in WTFC domain, WTFC should react normally by calculating the optimal packet rate and window. When congestion arises in TCP domain, proxy TCP sender should reduce its sending rate as a reaction to packet losses. However, WTFC sender continues to use its current congestion window and WTFC-TCP proxy buffer tends to overflow.

All three ACK loopback policies are considered in this case: far-end, near-end and proxy buffer loopback.

Further, proxy TCP sender can be a full or modified TCP sender. Near-end ACK loopback requires a full TCP sender to be used, to let it compete with other TCP flows in TCP domain. Far-end ACK loopback provides some feedback for WTFC sender from TCP domain as well as WTFC domain, so modified proxy TCP sender can be used.

Far-end ACK policy was at first considered with proxy TCP forwarder, a simple process that sends whatever packet is received at the WTFC-TCP proxy buffer. The performance of this combination was unpredictable, because both path capacity estimation and load estimation mechanisms of WTFC were influenced by TCP domain network load. This resulted in poor fairness, where WTFC used less than its fair share when congestion in TCP domain was low, and more than its fair share during heavy congestion in TCP domain.

To compensate for the TCP domain influence, TCP forwarder was then replaced with the full TCP sender. This had further effects on WTFC estimation processes due to the proxy TCP sender slow start mechanism.

Finally, a variant of TCP sender without initial slow start phase was included, but acceptable performance could not always be obtained.

As expected, near-end ACK loopback suffered from lack of flow control between WTFC and TCP domains. In this case, full proxy TCP sender was the only choice. Obviously, some signaling from TCP to WTFC domain is indispensable.

At this point, the proxy buffer ACK loopback was considered, with full TCP sender. Proxy WTFC receiver sends ACK to proxy WTFC sender after a packet is removed from WTFC-TCP proxy buffer. This way, if packets are accumulated at the WTFC-TCP proxy buffer, flow of WTFC acknowledgments is delayed, thus signaling possible congestion in TCP domain.

However, early simulations did not achieve acceptable performance. The reason was the proxy TCP sender algorithm that builds up its *cwnd* despite the constrained number of packets available from WTFC domain. When congestion in TCP domain finally occurs, proxy TCP sender continues to transfer incoming WTFC packets as allowed by high *cwnd*, despite the lack of TCP acknowledgments.

To limit *cwnd* build up on proxy TCP sender, a simple limiting algorithm is introduced: if *cwnd* tends to excess the current window (number of packets in flight) plus packets in the WTFC-TCP proxy buffer, *cwnd* is limited to that value plus some additive value (5 packets in simulations performed).

The combination of the proxy buffer ACK loopback with *cwnd* limiting policy provided acceptable performance in the simulation environment.

V. SIMULATIONS

Simulations were conducted using ns2 simulator (version 2.34) on the simplistic topology with seven nodes, node n2 being WTFC-TCP proxy, shown in Figure 3. Link capacity of 10 Mb/s between nodes n2 and n3 emulates a possible ADSL connection to the Internet which is often of restricted capacity. 10 Mb/s capacity between nodes n2 and n1 emulates often scarce resources in parts of an Internet provider network. In order to present the induced packet loss algorithm, Figure 6 was taken from a simulation conducted on the same topology, but with 100 Mb/s capacity between nodes n1 and n2.

In simulations presented, a flow under observation between nodes n0 and n4 starts at t=0s and ends at t=16s, which is the duration of simulation. In Section A, results for a TCP to WTFC domain flow from node n0 to node n4 are presented, and in Section B, results for a WTFC to TCP domain flow from node n4 to node n0. From t=2s to t=6s ten competing TCP Reno flows are active in TCP domain between nodes n0 and n2, and from t=10s to t=14s ten competing WTFC flows are active in WTFC domain between nodes n2 and n4. Competing flows use the same direction as the flow observed. Drop Tail algorithm is used on links in the WTFC domain and ARED (Adaptive RED) on links in the TCP domain.

To examine the effects of WTFC-TCP flow control proxy on QoS sensitive applications additional simulations were conducted on the same topology. In those simulations a UDP/CBR (User Datagram Protocol / Constant Bit Rate) flow is active in the WTFC domain between nodes n6 and n5 from t=1s to t=16s. The rate used is 64 kb/s, which is the VoIP standard. The UDP/CBR flow is placed in the WTFC domain rather than in the TCP domain, so as not to let TCP influence it. The assumption here is that QoS would be otherwise provided (enforced) in the provider network by

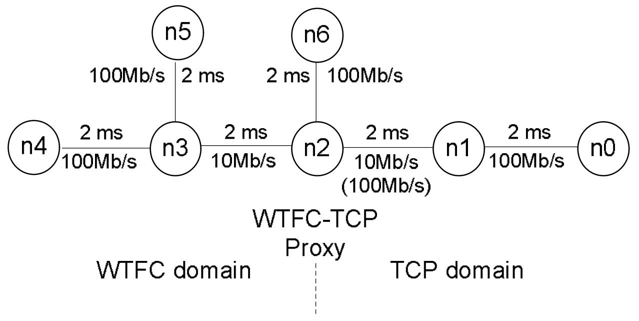

Figure 3. Simulation network topology.

Traffic Engineering (TE) under DiffServ paradigm.

A. TCP to WTFC domain flows

Packet flow dynamics for a TCP to WTFC domain flow with the induced packet loss algorithm on TCP-WTFC proxy buffer is shown in Figure 4. Slow start algorithm of the TCP sender causes losses and disrupted flow of packets in the first second of simulation time. In time intervals when the competing flows are active the slope of the curve, representing throughput, is reduced as expected. Calculated throughput is shown in Figure 9 and discussed in Section C.

Window for TCP sender and proxy WTFC sender, as well as TCP-WTFC proxy buffer occupancy are shown in Figure 5. The value of the TCP sender window going to 240 packets at the beginning of the simulation is not shown in the figure in order to limit the scale of the y-axis. TCP sender window follows the additive increase-multiplicative decrease (AIMD) rule of the TCP Reno algorithm, while proxy WTFC sender keeps its window steady according to the calculated optimal working point. Low TCP-WTFC proxy buffer occupancy shows the efficiency of the applied algorithm.

Packet dynamics in the event of an induced packet loss is

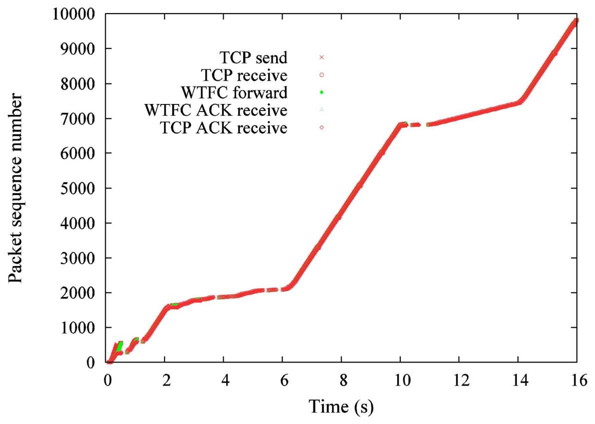

Figure 4. TCP to WTFC domain flow packet dynamics.

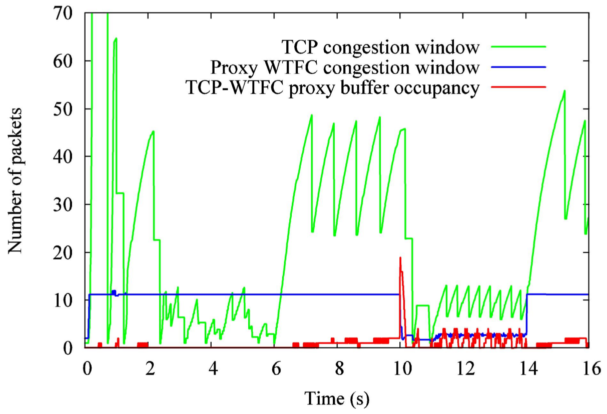

Figure 5. TCP to WTFC domain flow windows and proxy buffer occupancy.

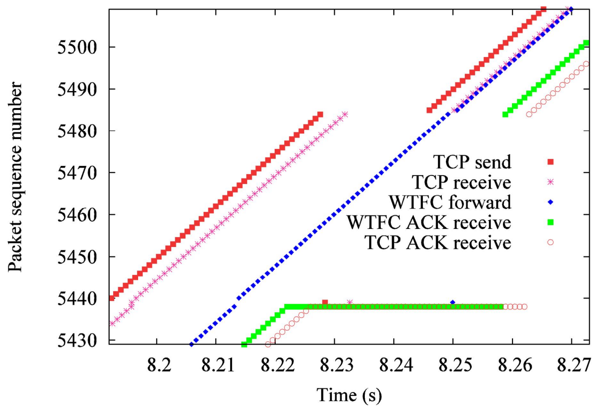

Figure 6. Packet dynamics in the event of induced packet loss.

shown in Figure 6. The increasing latency between TCP receive event and WTFC forward event on the proxy is due to the increasing proxy buffer occupancy, eventually resulting in induced packet loss and duplicated ACKs. After the packet loss, the TCP window is halved by the TCP Reno flow control algorithm. Consequently, there is a pause in sending new packets. TCP-WTFC proxy buffer deque, regulated by Proxy WTFC sender, remains steady.

B. WTFC to TCP domain flows

Packet flow dynamics for a WTFC to TCP domain flow with WTFC-TCP proxy buffer ACK loopback and the proxy TCP sender *cwnd* limiting algorithm is shown in Figure 7. Performance observed is similar to that of Figure 4. The first slope is smoother then in Figure 4 because WTFC sender does not use the aggressive slow start algorithm. Optimal working point is computed upon the arrival of the first ACK.

Window for WTFC sender and TCP proxy sender, as well as WTFC-TCP proxy buffer occupancy are shown in Figure 8. Low WTFC-TCP proxy buffer occupancy shows the efficiency of the applied algorithm. Proxy TCP sender window growth is limited to the number of packets in flight plus the number of packets in proxy buffer plus 5. Therefore, proxy TCP sender window is steady and following the change of WTFC window when there is no congestion in

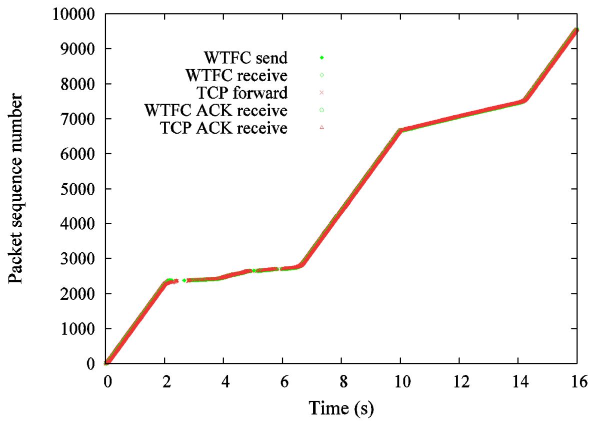

Figure 7. WTFC to TCP domain flow packet dynamics.

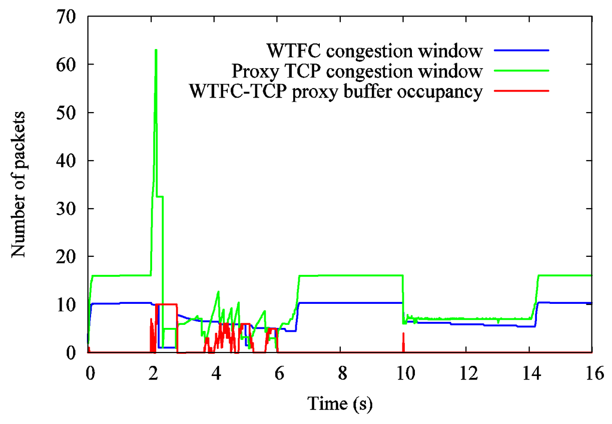

Figure 8. WTFC to TCP domain flow windows and proxy buffer occupancy.

TCP domain. Otherwise it follows the TCP Reno AIMD rule. The *cwnd* spike at t=2s shows that *cwnd* limiting algorithm should be improved.

C. *Throughput, delay and jitter*

Simulation results presented in this section were obtained with TCP to WTFC domain flows. Results with WTFC to TCP domain flows are similar. Comparison is made with results obtained when traditional TCP flow control is deployed on the entire network. Topology used is the same as with the WTFC-TCP proxy. Competing flows in this TCP-all network are TCP Reno flows in both the time interval from t=2s to t=6s and the time interval from t=10s to t=14s. Additional simulations were conducted on the TCP-all network with ARED algorithm on all links.

Throughput achieved by the TCP to WTFC domain flow under observation is comparable to that of a standard TCP Reno flow on the same topology, as well as on the topology with only ARED; see Figure 9. It shows good channel utilization when there is no competing traffic and fair reaction when competing flows are present.

The property of WTFC to estimate the optimal working point and to keep on average empty node buffers is expected to result in some QoS metric improvements when using WTFC to TCP flow control proxy, in comparison with results using only traditional TCP flow control. For verification, delay and jitter of VoIP traffic, represented by a UDP/CBR flow between node n6 and node n5, are observed.

As expected, delay when using the WTFC-TCP flow control proxy is generally smaller than delay on the equivalent TCP-all network, both with just ARED and with the combination of Drop Tail and ARED algorithms; see Figure 10. Peak delay with the proxy is 60 ms and without the proxy about 125 ms. While competing flows between nodes n2 and n4 are active, delay with the proxy is stabilized below 30 ms. In the same interval, delay of the TCP flow with ARED has almost 10 times greater jitter and the delay of the TCP flow with ARED and Drop Tail is near the peak value.

Jitter estimator MAPDV2 according to ITU-T G.1020 Recommendation is used to compute the jitter. Results are shown in Figure 11. Peak jitter value with the WTFC-TCP flow control proxy is 20 ms, while the same value for TCP-all flows exceeds 100 ms. The proxy reduces jitter in general as well as the fluctuation of jitter values. While competing

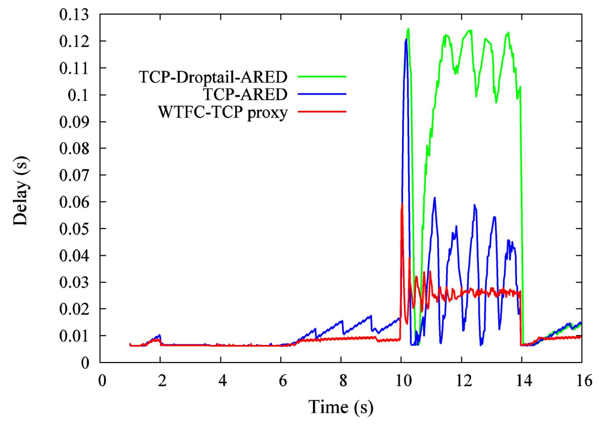

Figure 10. Delay of a CBR flow using WTFC-TCP flow control proxy and using standard TCP flow control.

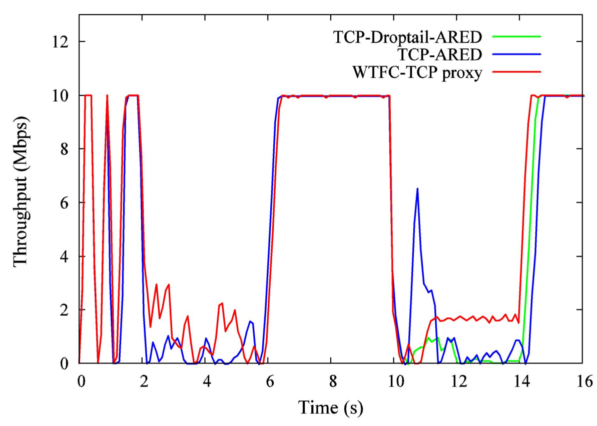

Figure 9. Throughput using WTFC-TCP flow control proxy and using standard TCP flow control.

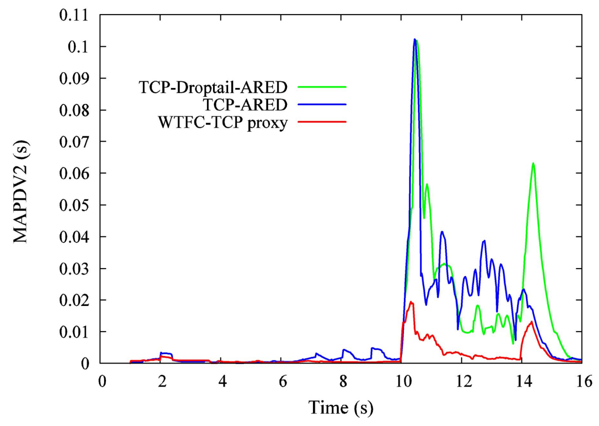

Figure 11. Jitter (MAPDV2) of a CBR flow using WTFC-TCP flow control proxy and using standard TCP flow control.

flows in WTFC domain are active, jitter drops from peak value to below 5 ms.

The UDP/CBR flow passes through the link between nodes n2 and n3 together with the TCP to WTFC domain flow and competing WTFC flows, or together with equivalent TCP flows. Both delay and jitter have the lowest values in the time interval from t=10s to t=14s when competing flows in TCP domain are active. Congestion is then present in the TCP domain, thus leaving the link between nodes n2 and n3 unsaturated.

VI. Conclusion and Future Work

In this paper, the concept of resolving the QoS at the edge between access network and the Internet is explored using WTFC flow control on the access network and standard TCP on the Internet. The WTFC-TCP flow control proxy is introduced in order to exploit the ability of WTFC to estimate the optimal working point for the available capacity and to keep on average empty node buffers.

A basic proxy model is defined, consisting of packet queues in both WTFC and TCP domains. WTFC and TCP flows are interconnected through proxy buffers located on the proxy node.

In this scenario main flow control problems arise when congestion is present on the receiver side of the model. In this case some kind of flow control signalization is needed between WTFC and TCP domains.

When a flow passes from TCP to WTFC domain, WTFC congestion is signaled to TCP sender by induced single packet loss. Proxy deliberately discards a packet when the TCP to WTFC proxy buffer occupancy exceeds double value of proxy WTFC sender congestion window. This way, the TCP sender *cwnd* is halved. Simulation results have shown that this simple mechanism operates efficiently.

In a scenario when a flow passes from WTFC to TCP domain, congestion notification from TCP domain to WTFC sender is obtained using delayed local WTFC acknowledgments. These are sent at the moment when a packet is taken from the WTFC to TCP proxy buffer. Additional proxy TCP sender *cwnd* limiting algorithm is used to prevent TCP from unnecessarily increasing *cwnd* during congestion in WTFC domain. Simulation results have shown that these two mechanisms combined provide satisfactory performance.

Simulation results presented make the WTFC-TCP flow control proxy concept promising for further research. Throughput achieved is comparable to that of a TCP-all network. Delay and jitter of a VoIP flow passing through the WTFC domain were taken in consideration as QoS metrics. They show considerable improvement over delay and jitter on a TCP-all network. Future work includes expanding algorithms with more sophisticated features as well as conducting experiments on more complex topologies and in wider condition range. Finally, the QoS for multimedia flows in integrated environment with data traffic needs to be extensively verified.

References

[1] A. Bakre and B. R. Badrinath, “I-tcp: Indirect tcp for mobile hosts,” in 15th International Conference on Distributed Computing Systems, pp. 136–143, 1995.

[2] H. Balakrishnan, V. N. Padmanabhan, G. Fairhurst and M. Sooriyabandara, “TCP Performance Implications of Network Path Asymmetry,” IETF RFC 3449, Dec 2002.

[3] H. Balakrishnan, S. Seshan, E. Amir and R. H. Katz, “Improving tcp/ip performance over wireless networks,” in 1st Annual International Conference on Mobile Computing and Networking, New York, NY, USA, 1995, pp. 2–11, ACM.

[4] J. Border, M. Kojo, J. Griner, G. Montenegro and Z. Shelby, “Performance enhancing proxies intended to mitigate link-related degradations,” IETF RFC 3135, June 2001.

[5] R. Chakravorty, S. Katti, I. Pratt and J. Crowcroft, “Using TCP flow-aggregation to enhance data experience of cellular wireless user,” IEEE Journal on Selected Areas in Communications, vol. 23, no. 6, pp. 1190-1204, June 2005.

[6] S. Cheng, C. Guo, J. Li and L. Zhu, “Design and Analysis of an XCP-TCP Gateway,” International Conference on Networking 2008 (ICOIN 2008), pp.1-5, Jan. 2008, doi: 10.1109/ICOIN.2008.4472746.

[7] D. Dutta and Y. Zhang, “An Active Proxy Based Architecture for TCP in Heterogeneous Variable Bandwidth Networks,” Proc. IEEE GLOBECOM 2001, pp. 2316 – 2320, November 2001.

[8] J. V. Jacobson, “Congestion Avoidance and Control,” Proc. ACM SIGCOMM ’88, vol. 18, no.4, pp. 314-329, Stanford, USA, August 1988.

[9] D. Murray, T. Koziniec and M. Dixon, “Solving Ack Inefficiencies in 802.11 Networks,” Proceedings of the 3rd IEEE International Conference on Internet Multimedia Services Architecture and Applications IMSAA'09, pp. 1-6, 2009.

[10] J. Ozegovic, “Filtering schemes for the window-time space flow control (WTFC),” Proc. ICCCN'00, pp. 575 - 580, Las Vegas 2000, USA.

[11] J. Ozegovic, “Implementation algorithms for the window-time space flow control (WTFC),” Proc. ICCCN'99, pp. 30 - 35, Boston 1999, USA.

[12] J. Ozegovic, “Window-time space flow control (WTFC),” Proc. ICCCN98, pp. 800 - 807, Lafayette 1998, USA.

[13] H. Shimonishi, T. Hama and T. Murase, “Tcp-adaptive reno for improving efficiency-friendliness tradeoffs of tcp congestion control algorithm,” PFLDnet, pp. 87–91, Feb. 2006.

[14] D. Velenis, D. Kalogeras and B. Maglaris, “SaTPEP: a TCP Performance Enhancing Proxy for Satellite Links,” Proc. 2nd International IFIP-TC6 Networking Conference on Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; and Mobile and Wireless Communications (NETWORKING '02), pp. 1233-1238, 2002.