AN OPENWRT SOLUTION FOR FUTURE WIRELESS HOMES

Claudio E. Palazzi, Matteo Brunati

Dipartimento di Matematica Pura e Applicata
Università degli Studi di Padova
Padova, Italy
Email: cpalazzi@math.unipd.it

Marco Roccetti

Dipartimento di Scienze dell'Informazione
Università degli Studi di Bologna
Bologna, Italy
Email: roccetti@cs.unibo.it

ABSTRACT

Most of future digital services for home and office users will be deployed and delivered through the Internet and wireless connectivity. However, employing regular access points (APs) and protocols will not allow an efficient coexistence among heterogeneous application flows. Indeed, real-time and elastic applications are (and will be) supported by different protocols and featured with different performance requirements: low per-packet delay for the former and high throughput for the latter. In this work, we present a work-in-progress evaluation of a new AP prototype able to guarantee a fast and smooth data delivery for real-time streams while maintaining a high throughput for TCP-based applications. Our approach is based on a user-space modification of the AP's code based on OpenWRT operating system so as to appropriately shape transiting network traffic. Preliminary experimental results confirm that our solution represents an optimal candidate to become the center of future wireless in-home scenarios.

Keywords— OpenWRT, Smart Access Point, Testbed, Wireless Protocols.

1. INTRODUCTION

It is easy to foresee that homes and SOHOs (Small Office/Home Office) in the next future will make extensive use of wireless connectivity to support a broad range of devices such as personal computers, consoles, TV sets, and other appliances. File downloading or sharing, Internet browsing, video or audio streaming, VoIP, and online gaming are just a few, even if representative, examples of connectivity-based multimedia applications that will be simultaneously run by users, [1, 2, 3, 4, 5, 6, 7, 8].

Unfortunately, the current TCP/IP suite of Internet protocols has been developed tens of years ago and presents several issues when plunged into wireless, heterogeneous scenarios. First, we have to remember that real-time services are usually based upon the UDP transport protocol, whereas elastic ones make use of TCP (which implements flow control functionalities). When heterogeneous applications coexist, so do UDP and TCP, unfortunately not in a friendly way. Indeed, TCP's congestion control mechanism is designed to continuously probe the channel for more and more bandwidth until saturating the channel and the buffer at the bottleneck; at that point, some packet is lost and the transmission speed is halved before starting again with the same probing pattern. This behavior can negatively affect real-time applications as it creates queues (at the bottleneck buffer), thus increasing the per-packet delivery latency, which is the main performance requirement for real-time, interactive applications.

This result, even if predictable, is quite surprising as TCP is generally considered more friendly than UDP, since only the former has a congestion control functionality. Indeed, citing from one of the most widely adopted textbook for networking classes [9]: *"Although commonly done today, running multimedia applications over UDP is controversial to say the least. [...] the lack of congestion control in UDP can result in high loss rates between a UDP sender and receiver, and the crowding out of TCP sessions - a potentially serious problem"*.

Even if this statement is still true when the available bandwidth is scarce, the larger broadband connectivity offered today (and in future) may overturn this situation. As a result, if one user is engaged with an interactive online game while another user, sharing the same bottleneck, is downloading a big file, the former frequently notices a lack of responsiveness in receiving game updates and in executing game actions; the game will result less fun as the user will not be able to compete in a fair way [2, 10, 11, 12].

Instead, wireless connectivity should fairly support any kind of application, providing both high downloading throughput and low per-packet delays, which are the main performance metrics for elastic and real-time applications, respectively. To this aim, we propose a solution that makes use of enhanced APs, standard protocol features, and available information on the on-going traffic. More in detail, our solution is based on the *OpenWRT* operating system [13] and makes the AP able to modify on-the-fly the advertised window of each transiting TCP ACK packet. This way, the transfer rate of TCP flows is limited so as to not exceed the available bandwidth.

Indeed, through an appropriate limitation of the maximum transfer rate for each flow, downloading applications produce a smooth traffic that efficiently utilizes the available channel without incurring in congestion losses that would degrade their performances and, at the same time, does not create queues at the

AP that would increase per-packet delays of real-time applications.

To demonstrate our claims, we have implemented a prototype of our solution and performed preliminary evaluation. Results gathered on the field and reported in this paper are very encouraging and demand for further investigation.

The rest of the paper is organized as follows. In Section 2, we discuss our proposed solution. The testbed assessment is presented in Section 3, while experimental results are shown in Section 4. Finally, Section 5 concludes this paper also indicating future directions for this work.

2. PROPOSED SOLUTION

We are aiming at finding the best solution to the tradeoff relationship existing between TCP throughput and real-time application delays. We exploit the advertised window field regularly present in all TCP ACK packets to limit the bandwidth utilized by each TCP flow. Indeed, the actual sending rate of a TCP flow depends on its current congestion window and on the advertised window decided by the receiver: the sending windows is determined as the minimum between these two values [9]. It is hence evident how the advertised window perfectly embodies a natural upper bound for the sending rate of TCP flows. Our contribution is that of adding intelligence to the AP so as to making it able to appropriately modify on-the-fly the advertised window field of transiting TCP ACKs. All flows generated by and for wireless nodes located inside the house will have to pass through the AP, thus making it the most suitable place for implementing our solution. Since its properties, this solution has been named *Smart Access Point with Low Advertised Window* (*SAP-LAW*).

Limiting the sending window could guarantee the same or even a higher throughput with respect to utilizing regular TCP. Indeed, an appropriate limitation of the advertised window can generate a more stable transmission rate at steady state: something more similar to the behavior of a TCP Vegas' sending window than to the saw-tooth shaped sending window of legacy TCP versions. At the same time, queuing delays will be avoided, thus producing benefits even for time sensitive flows such as those generated by real-time applications.

An important component of SAP-LAW is certainly represented by the formula utilized to compute the advertised window (i.e., the maximum transmission rate) for each TCP-based flow. Clearly, a static value would not be able to address different networking configurations as at a certain point there may be a different amount of UDP-based traffic and a different number of TCP-based connections. Therefore, the advertised window value for each transiting ACK has to be dynamically computed taking into account several factors.

It has been demonstrated in scientific literature that the bottleneck of a connection is typically at the last mile link [14]. The bandwidth of this link can be easily known as it corresponds to the connectivity that the user is paying for; alternatively, software such as *CapProbe* may be employed [15]. Exploiting this information, (1) and (2) are two possible formulas to compute the appropriate advertised window for the various TCP-based flows at a certain time *t*.

$$maxTCPrate(t) = \frac{(C - UDPTraffic(t))}{numberTCPflows(t)} \quad (1)$$

$$maxTCPrate_i(t) = \frac{(C - UDPtraffic(t)) * RTT_i}{\sum \frac{RTT_i}{avg\_RTT}} \quad (2)$$

In (1) and (2), *C* is the capacity of the bottleneck. The first formula does not take into account the RTT-unfairness problem of TCP flows [16, 17] and simply determine the *maxTCPrate* at time *t* as the ratio between the bottleneck capacity left free by the UDP flows and the number of concurrent TCP flows. Therefore, all TCP flows will be subject to the same limited advertised window.

Instead, (2) computes different *maxTCPrate* values: one for each TCP flow *i*, depending on its actual RTT (RTT_i). In essence, the computed advertised windows are in inverse proportion with the respective RTTs so as to foster a fair sharing of the available bandwidth (i.e., the same final throughput). This can be achieved at the expenses of monitoring the RTTs of the various concurrent TCP flows and computing their average (avg_RTT).

The SAP-LAW approach is also in accordance with other proposals available in literature such as, for instance, [18]. However, whereas [18] requires modifications at both the AP and the receiver, our scheme exploits only an enhanced AP.

In the following subsection we provide more details on implementation issues related to our solution, both from a hardware and a software standpoint.

2.1. Implementation Details

For the hardware part we looked for an AP that could be modified without excessive burden and that represented an off-the-shelf product; we have hence chosen NETGEAR WGT634U [19, 20]. Furthermore, we did some preliminary tests also with possible alternatives such as, for instance, Fonera+ (Fonera version 2201).

We aimed at a solution that could easily be implemented in houses and small offices, running with a high number of different APs and embedded hardware. Therefore, we searched for a Linux based operating systems for embedded architectures, in order to potentially use our solution with a large number of APs. This choice allowed us to work on our PC in Linux environment and then port our software on the actual AP only when completed. The best choice for our purposes was OpenWRT: a Linux kernel based operating system for embedded solutions, primarily oriented for networking support [13].

Focusing on the software components, it should be said that the closer the solution to the hardware, the faster the execution. On the other hand, solutions implemented with a high hardware dependency are not easily portable on embedded platforms with different network interfaces. We have hence evaluated various

possible solutions such as modifying the network driver, modifying the existing kernel module, and adding a new kernel module [21, 22]. To allow the widest portability, we have then decided to implement a user-space solution reserving for future work to provide also a kernel-based solution. However, we have to say that for a realistic wireless home networking scenario, as the one considered in our experiments, our user-space solution showed no problem in terms of execution speed.

More in detail, the adopted solution is based on *Netfilter*, the Linux kernel networking framework. Netfilter permits to write modules that register themselves with the kernel; then, through specific tables that state how to handle network flow packets, the kernel is enabled to utilize these modules. Furthermore, Netfilter provides developers with APIs to interface with the kernel from user space. Clearly, this involves context switches and packets that have to be copied back and forth between kernel space and user space; on the other hand, nothing impedes to port the developed user-space module into the kernel in future.

As currently our solution has been developed as a SAP-LAW version for user-space, we have named it *SAP-LAW-US*, or simply *SLUS*. Our prototype is clearly a proof-of-concept and we used it to create a testbed able to demonstrate its basic functionalities and benefits achievable.

More in detail, SLUS implementation is composed by two main parts: the first one is the definition of *iptables* rules so as to forward TCP ACK packets to a specific user-space queue. The second part of the implementation is made by an application written in C language that reads from the aforementioned queue via *libnetfilter-queue* (the Netfilter API for reading and modifying packets at the user-space layer). This second part is devoted to intercept and modify on-the-fly the advertised window field of transiting TCP ACK packets. In the current version of our SLUS AP prototype, the advertised window of TCP ACK packets can be modified with a fixed value or through (1); we are now working also on the implementation of (2).

3. TESTBED ASSESSMENT

We have tried to recreate a realistic scenario in order to study SLUS in a real network with both UDP and TCP flows, with RTTs of tens of ms. The testbed we deployed to demonstrate SLUS' efficacy is shown in Fig. 1.

It is important to note how the testbed can be easily extended to all experiments test cases which needs an AP point that interconnect a client and a server.

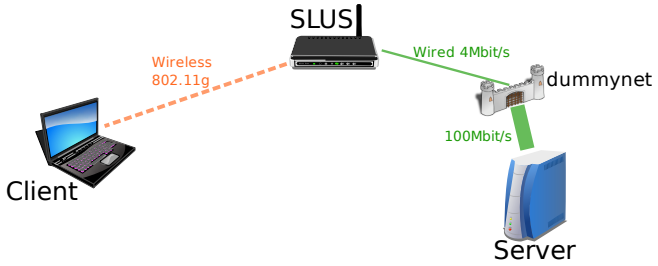

Fig. 1. Testbed configuration.

Table 1. Traffic flows generated by ITGSend (D-ITG suite).

Flow type	Protocols	Start time	End time	RTT
CBR flow	UDP	0s	180s	60ms
Online Game	UDP	45s	180s	60ms
VoIP	UDP	90s	180s	60ms
FTP flow	TCP	135s	180s	60ms

As anticipated, for the wireless network we employed OpenWRT (including our SLUS solution) on a NETGEAR WGT634U AP. Instead, to emulate Internet real-time and elastic services we have used two notebook PCs. One PC was the server of the communications and generated UDP and TCP traffic flows. The other notebook was the client, which received the traffic generated by the server and collected all the flow statistical information used to draw graphs presented in Section 4. The server was connected to the AP; between the two, the Internet is emulated through the *Dummynet* emulator. Clients (one for each application run) were connected to the AP via the IEEE 802.11g protocol, with no encryption. The bottleneck was located between the Dummynet and the AP, and corresponded to 4 Mbit/s.

We have used various software to configure and analyze experiments. In particular, Dummynet was used to emulate the Internet connection of the network; *D-ITG* supplied network traffic emulation generating both UDP- and TCP-based flows; *Tcpdump* and *Tcptrace* were utilized to collect TCP traffic information for a post-processing graphs production.

In Table 1 we describe the traffic flows generated for our tests through the *ITGSend* program of the D-ITG suite. In each experiment, we had three real-time flows based on UDP and one elastic flow based on TCP; the four flows started at different times, with the TCP flow starting last, after 135s of simulation. The first flow is a simple constant bit rate (CBR) flow, with 200 packets of 1 KB each sent every second. The second one corresponds to the network activity of a *Counter Strike* (*CS*) online game session, with game packets of 200 B sent from the server to the client every 40 ms. The third flow regards a VoIP call with the G.711.1 Codec over the RTP protocol. Cumulatively, these three UDP-based flow consumes a quantity of bandwidth that oscillates around 1.7 Mbit/s thus leaving around 2.3 Mbit/s available. Finally, the forth flow, the TCP-based one, corresponds to a long FTP downloading session where each packet has a payload of 1 KB. The closer this FTP flow gets to consume 2.3 Mbit/s of bandwidth, the higher the efficiency.

4. EXPERIMENTS AND RESULTS

To validate SLUS implementation and verify its applicability to the real world, we used the testbed described in the previous section. Different testbed configurations have been tried. For the sake of conciseness we report here results for only one significative case which considers an RTT of 60 ms. Nonetheless, results for other configurations are coherent with the one

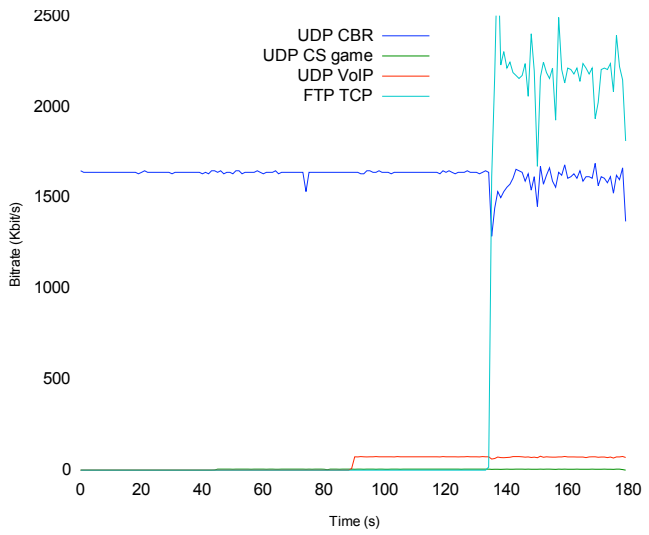

Fig. 2. FTP/TCP flow bitrate without SLUS application; average bitrate 2.183 Mbit/s.

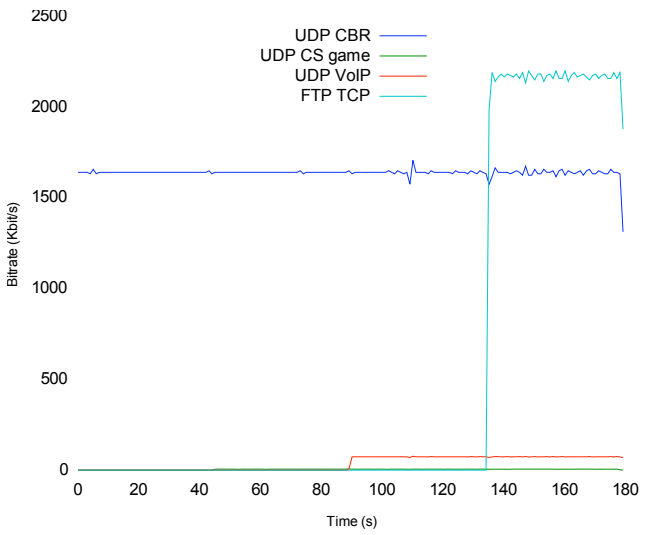

Fig. 3. FTP/TCP flow bitrate running SLUS on the AP; average bitrate 2.164 Mbit/s.

reported here and their charts can be found on the project web site [23].

In particular, Fig. 2 and Fig. 3 show the generated FTP bitrate for the regular configuration or when applying SLUS, respectively. Whit SLUS, the advertised window is set equal to an appropriate value so as to consume almost all the bandwidth available without generating queues; this value corresponds to the result of (1). However, since in this configuration there is only one TCP-based flow, (1) and (2) would generate exactly the same outcome. As evident from the two charts, SLUS enables a smooth bit rate that does not affect significantly the final average throughput (2.183 Mbit/s vs. 2.164 Mbit/s). In both cases, the bandwidth utilization with respect to the available one (around 2.3 Mbit/s) is very high.

Since we have demonstrated that the adopted SLUS configuration does not harm the throughput of elastic flows, we have now to evaluate if it is also able to fulfil its promise of ensuring low per-packet delays to real-time flows. We have hence monitored the packet inter-arrival time of the concurrent online game flow. As the utilized CS game emulator regularly generate at server side a game packet every 40 ms, with no queue we expect an inter-arrival time of 40 ms for the same packets at client side. Instead, to longer queues at the bottleneck buffer will correspond wider oscillations of the packet inter-arrival time.

In point of this, Fig. 4 and Fig. 5 show how the inter-arrival time of the UDP-based online game flow changes when passing from a regular configuration to employing SLUS. Clearly, SLUS sensibly reduces the oscillations of the inter-arrival time of online game packets, thanks to a smoother traffic progression in the network that avoids packet queuing.

Since SLUS avoids buffer exploitation by TCP-based flows, these flows experience much less packet loss or no loss at all. Indeed, Fig. 6 reports the packet loss experienced by the CS gaming flow in a regular network configuration. When the TCP flow is active, its continuous probing for more bandwidth eventually fills up the buffer at the AP thus causing packet losses. Instead, the benefits achievable through our solution are demonstrated also by the fact that no losses at all happened in our tests when implementing SLUS; we do not report any figure for this as it will simply result an empty chart.

5. CONCLUSION AND FUTURE WORK

This article reports on the crucial problem of facilitating the coexistence among heterogeneous multimedia flows in a future wireless home. In particular, we aimed with success at ensuring both low per-packet delay to real-time interactive applications and high throughput to elastic ones. Our solution, named SLUS, has been implemented in OpenWRT on a real AP thus creating a functioning prototype. We have hence been able to generate a realistic testbed that produced very encouraging results.

Furthermore, since we propose neither a new protocol nor a new architecture, our solution is also highly tolerable from a deployability standpoint. Indeed, we simply designed an enhanced AP with a re-engineered packet forwarding functionality. Customers have hence just to buy our AP or to install the appropriate patch on their APs to improve the ability of their wireless connectivity in handling heterogeneous traffic and applications; no modifications are required at server or client side.

Our work can be extended in several directions. First of all, we would like to compare the current user level solution with a kernel level one. Indeed a kernel module implementation could improve the efficiency of TCP flows modifications and the scalability of the system in terms of number of concurrent flows; this would allow to consider even more complex environment than a wireless house. We are planning to build our kernel implementation upon Netfilter. More in detail, as done for the user space version, even the kernel space one will be decomposed

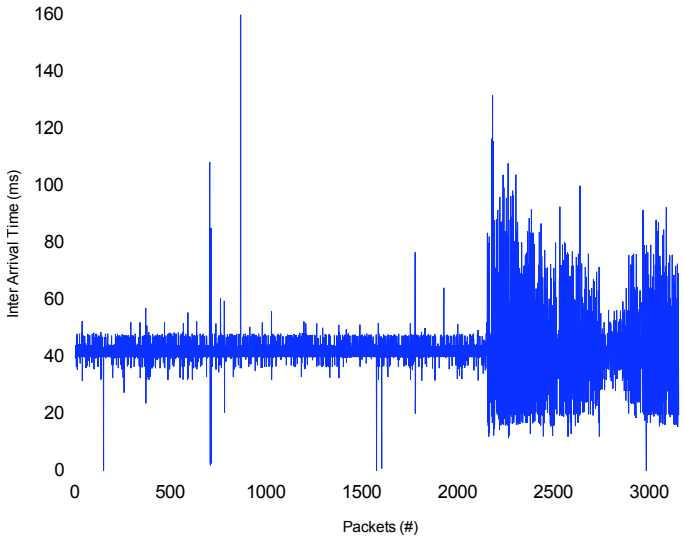

Fig. 4. Online game flow inter-arrival time; regular configuration without SLUS.

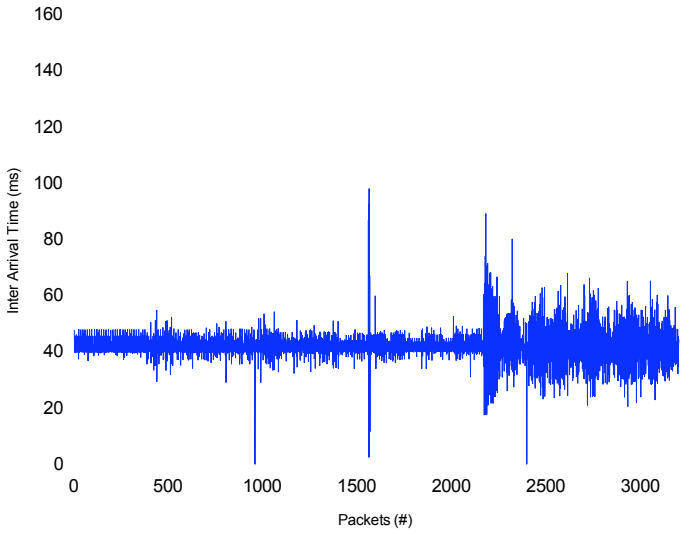

Fig. 5. Online game flow inter-arrival time; SLUS implemented.

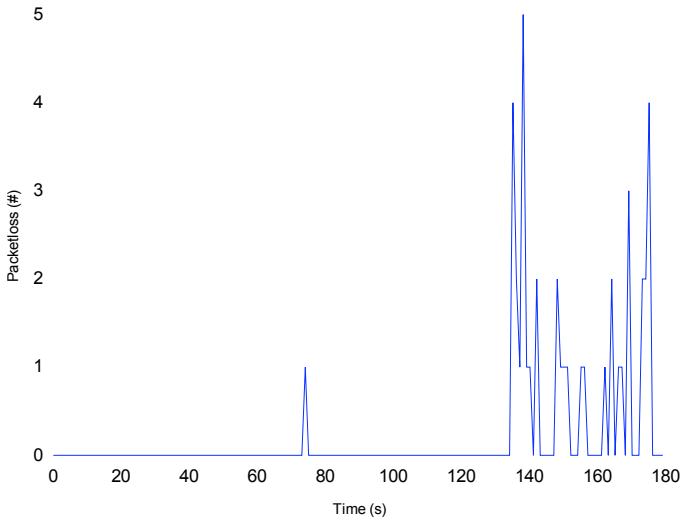

Fig. 6. Online game flow packet losses without SLUS; packet losses completely disappear with SLUS.

into two parts: the first one will be an iptables shared library adding a new command. In this way, we will have a user-space interface for the kernel module with a secure solution. The kernel module will be in the form of a *kernel hook* that register itself to the Linux kernel when invoked by the new iptables extension. This will permit kernel module efficiency with a user-space control.

Another significant feature to insert in SAP-LAW implementations, both the user-space one or the kernel module, could be an enhanced algorithm to compute the TCP advertised window in a more efficient and fair way. Also, the solution's code can be optimized to further boost the program execution.

Finally, since the complexity of possible scenarios, we plan to compare our current solution against possible (even if probably more complex) QoS alternatives in different networking conditions through various realistic scenarios [24]. We would also like to enrich our work with an analysis model framework able to formally compare the functionalities of alternative solutions and validate achieved testbed results [25, 26].

6. REFERENCES

[1] M. Furini, "Mobile Games: What to Expect in the Near Future", *Proc. of GAMEON 2007*, Bologna, Italy, Nov 2007.

[2] A. Ploss, S. Wichmann, F. Glinka, S. Gorlatch, "From a Single- to Multi-server Online Game: A Quake 3 Case Study Using RTF", *Proc. of ACM ACE 2008*, Yokohama, Japan, Dec 2008.

[3] A. Balk, D. Maggiorini, M. Gerla, M. Sanadidi, "Adaptive MPEG-4 Video Streaming with Bandwidth Estimation", *Proc. of 2nd QOS-IP 2003*, Milan, Italy, Feb 2003.

[4] A. Balk, Dario Maggiorini, M. Gerla, M. Sanadidi, "Adaptive MPEG-4 Video Streaming with Bandwidth Estimation", Special Issue of Computer Networks Journal, vol. 44, no. 4, Mar 2004, 415-439.

[5] S. Ferretti, "A Synchronization Protocol for Supporting Peer-to-Peer Multiplayer Online Games in Overlay Networks", *Proc. of the 2nd ACM DEBS08*, Rome, Italy, Jul 2008.

[6] S. Ferretti, "Cheating Detection through Game Time Modeling: A Better Way to Avoid Time Cheats in P2P MOGs?" Multimedia Tools and Applications, Springer, vol. 37, no. 3, May 2008, 339-363.

[7] V. Ghini, G. Pau, P. Salomoni, "Always Best Served Music Distribution for Nomadic Users over Heterogeneous Networks", IEEE Communication Magazine Entertainment Everywhere: System and Networking Issues in Emerging Network-Centric Entertainment Systems, vol. 43, no. 5, May 2005, 69-74.

[8] G. Marfia, P. Lutterotti, S. Eidenbenz, G. Pau, M. Gerla, "FairCast: Fair Multi-Media Streaming in Ad Hoc Networks through Local Congestion Control", *Proc. of the 11th ACM MSWIM*, Vancouver, Canada, Oct 2008.

[9] J. F. Kurose, K. W. Ross, *Computer Networking: A Top-Down Approach Featuring the Internet*, Addison Wesley Longman, Boston, MA, USA, 2001.

[10] L. Pantel, L. C. Wolf, "On the Impact of Delay on Real-Time Multiplayer Games", *Proc. of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, Miami, FL, USA, May 2002.

[11] C. E. Palazzi, S. Ferretti, M. Roccetti, G. Pau, M. Gerla, "What's in that Magic Box? The Home Entertainment Center's Special Protocol Potion, Revealed", IEEE Transactions on Consumer Electronics, vol. 52, no. 4, Nov 2006, 1280-1288.

[12] C. E. Palazzi, N. Stievano, M. Roccetti, "A Smart Access Point Solution for Heterogeneous Flows", *Proc. 2009 IEEE International Workshop on Ubiquitous Multimedia Systems and Applications (UMSA'09) - International Conference on Ultramodern Telecommunications (ICUMT 2009)*, St. Petersburg, Russia, Oct 2009.

[13] OpenWRT website: *http://openwrt.org/*

[14] H. Jiang, C. Dovrolis, "Why Is the Internet Traffic Bursty in Short (Sub-RTT) Time Scales?" *Proc. of ACM SIGMETRICS 2005*, Banff, AL, Canada, Jun 2005.

[15] R. Kapoor, L.-J. Chen, L. Lao, M. Gerla, M. Y. Sanadidi, "CapProbe: A Simple and Accurate Capacity Estimation Technique", ACM SIGCOMM 2004, Portland, OR, USA, Sep 2004.

[16] F. Kelly, "Fairness and stability of end-to-end congestion control", European Journal of Control, vol. 9, 2003, 159-176.

[17] C. Caini, R. Firrincieli, "TCP Hybla: a TCP Enhancement for Heterogeneous Networks", International Journal of Satellite Communications and Networking, John Wiley & Sons, Vol. 22 , No. 5, Sep 2004, 547-566.

[18] L. L. H. Andrew, S. V. Hanly, R. G. Mukhtar, "CLAMP: Active Queue Management at Wireless Access Points", *Proc. of the 11th European Wireless Conference 2005*, Cyprus, Apr 2005.

[19] OpenWRT wiki, *http://kb.netgear.com/app/products/model/a_id/2598*

[20] OpenWRT wiki, *http://wiki.openwrt.org/oldwiki/openwrtdocs/hardware/netgear/wgt634u*

[21] L. Deri, *High-speed dynamic packet filtering*, deri@ntop.org.

[22] E. Weigle, *Re: [ns] linux tcp/ip stack modifications*, *http://www.isi.edu/nsnam/archive/ns-users/webarch/2000/msg06081.html*, December 2000, ns mailing list.

[23] SAP-LAW user-space implementation experimental results document: *http://redmine.matteobrunati.net/attachments/download/28/Esperimenti2.pdf.zip*

[24] J. Tang, X. Zhang, "Cross-Layer-Model Based Adaptive Resource Allocation for Statistical QoS Guarantees in Mobile Wireless Networks", IEEE Transactions on Wireless Communications, Vol. 7, No. 6, Jun 2008, 2318-2328.

[25] M. Garetto, R. Lo Cigno, M. Meo, M. Ajmone Marsan, "Closed Queueing Networking Models of Interacting Long-Lived TCP Flows", IEEE/ACM Trans. on Networking, Vol.12, No.2, Apr 2004, 300-311.

[26] F. Corradini, R. Gorrieri, M. Roccetti, "Performance Preorder: Ordering Processes with Respect to Speed", *Proc. Mathematical Foundations of Computer Science*, LNCS n. 969, Springer Berlin, 1995, 444-453.