Multi-service link layer enhancements for the wireless Internet

George Xylomenos and George C. Polyzos
{xgeorge,polyzos}@aueb.gr
Mobile Multimedia Laboratory
Department of Informatics
Athens University of Economics and Business
Patision 76, Athens 104 34, Greece

***Abstract*— Although TCP and UDP application performance over wireless links may be substantially improved via link layer error recovery, different schemes are appropriate for each application class. We present a multi-service link layer architecture that simultaneously enhances the performance of diverse applications by supporting multiple error recovery mechanisms in parallel. We simulated concurrent file transfers and WWW browsing over TCP and continuous media distribution over UDP using our architecture. The results show that each application achieves similar improvements as when it operates alone over its preferred scheme, despite the parallel execution of diverse applications.**

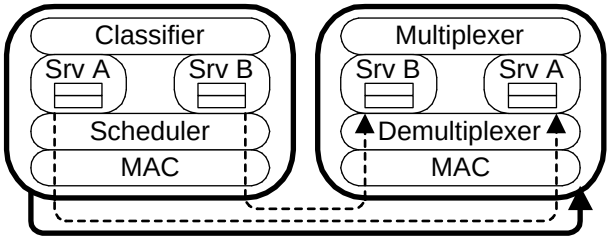

Fig. 1. Multi-service link layer architecture

I. Introduction

The growth of the Internet is only paralleled by the growth in wireless communications, especially in the areas of *Cellular Telephony* and *Wireless Local Area Networks* (WLANs). While the popularity of such wireless systems makes their integration into the Internet desirable, higher layers make assumptions about link performance that cannot be met by wireless links, which are slower and less reliable than wired ones. Thus, although supporting IP over such links is easy, the resulting performance is poor [1].

While UDP performance over wireless links has not been extensively studied, considerable work has been devoted to TCP. Most TCP enhancements try to avoid triggering congestion recovery due to wireless errors. One approach is to *split* TCP connections into a wireless and a wired part [2], so as to perform recovery over the wireless part only. Other schemes *freeze* TCP state when persistent errors are detected [3], so as to avoid invoking congestion control.

Alternatively, error recovery can be performed locally at the link layer, as in Cellular *Radio Link Protocols* (RLPs) [4]. RLPs use a single scheme which may be inappropriate for some traffic and they may interfere with TCP [5], leading to conflicting TCP and link layer retransmissions. Another local approach is to *snoop* inside TCP streams at the wireless base station so as to transparently retransmit lost segments when duplicate ACKs arrive [6]. This fails with IP security which hides the TCP header.

We have previously studied the performance of file transfer over TCP and continuous media distribution over UDP with various link layer schemes [7]. Our results showed that both TCP and UDP performance can be considerably improved, but that each application class preferred different approaches. While TCP applications favored mostly reliable error recovery, the delay sensitive UDP application favored less reliable but lower delay solutions.

We present below a multi-service link layer approach for improving the application performance over wireless links. Section II presents our multi-service link layer architecture. Section III describes our simulation setup. We then discuss the performance of three applications executing in parallel: Section IV covers file transfer, Section V covers World Wide Web browsing and Section VI covers continuous media distribution. Section VII summarizes our conclusions.

II. Multi-Service Link Layer Architecture

In order to support diverse Internet applications we have designed a *multi-service link layer* architecture supporting multiple enhancement services over a single physical link. With one service per transport protocol we would have to bundle together different applications, while with one service per application we would have to provide too many services. We therefore used one service per *application class*, that is, a group of applications with similar requirements. All TCP applications belong to the same class, as TCP fully controls their network behavior. For UDP applications, requirements may vary widely. Continuous media distribution belongs to an error tolerant delay sensitive class.

Fig. 1 outlines our multi-service architecture, showing data flow in one direction. Network layer packets are passed to services based on their application class. Since higher layers are unaware of the multi-service concept, a classifier is used for this task. A heuristic classification scheme is to check the IP

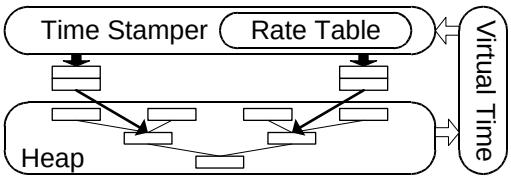

Fig. 2. Self clocked fair queueing scheduler

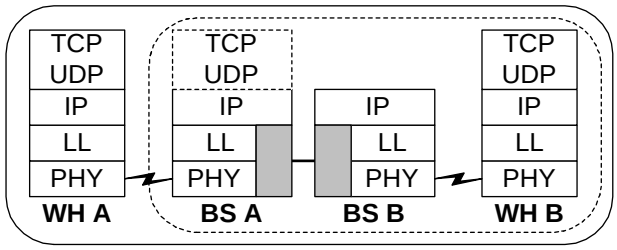

Fig. 3. Simulation topologies

protocol field to distinguish between TCP and UDP, and the TCP/UDP *port* field to detect applications [7]. When the *Differentiated Services* architecture [8] is used, we can exploit the IP DS field, which is visible even with IP security, unlike the protocol and port fields. Packets that cannot be matched to any service are mapped to a default service providing no error recovery. Services can use any mechanisms they desire and their parameters may be optimized for the underlying wireless link.

All services eventually pass their outgoing frames to the scheduler, which tags each frame with a service number and passes it to the MAC sublayer for transmission. At the receiver, frames are passed by the MAC sublayer to a demultiplexer which distributes them to services based on their tags. The services may eventually release packets to the multiplexer which passes them to the network layer. Thus, the peer services communicate over *virtual links*, multiplexed over a single physical link. New services may be added by simply inserting a module at both ends of the link. Services have single entry and exit points, thus allowing existing link layer code to be used without modifications.

Since services may inflate their data streams with arbitrary error recovery overhead, we have introduced a frame scheduler to ensure that the link will be *impartially* shared. Impartiality means that each service should receive the same amount of bandwidth as in a single service system. Assume that over a period of time the classifier allocates a_i bytes of traffic to service i. With a single service, all these data would be transmitted over that period, thus service i would receive a fraction $f_i = a_i / \sum_{j=1}^{n} a_j$ of the total bandwidth. The multi-service system is impartial if it allocates this exact fraction f_i to service i. A service performing no error recovery will get the same data bandwidth as with a single service. An error recovery service will have to split this bandwidth between its data and its overhead.

Impartiality is achieved by a *Self Clocked Fair Queueing* (SCFQ) scheduler [9], which enforces the desired bandwidth allocations when the link is loaded. When a service is idle, its bandwidth is proportionately shared among the rest. Fig. 2 gives an outline of the scheduler. The rate table holds the bandwidth fraction allocated to each service. The scheduler maintains a *virtual time*, initially 0, which is equal to the *time stamp* of the last packet transmitted. Frames submitted to the scheduler are kept in separate FIFO queues per service. To determine the time stamp of an incoming packet, we divide its size by its service rate and add it to the time stamp of the previous frame in its queue. If the queue is empty, we add it to the current virtual time. When the link is idle, the frame with the *lowest* virtual time is dequeued, the virtual time is updated and the frame is transmitted.

Since the time stamps in each queue are increasing, the scheduler keeps all queues in a sorted *heap*, based on their first frame, thus allowing immediate selection of the next frame to send. The heap is always sorted when a frame is removed for transmission. If the queue becomes empty, it is removed from the heap. When an empty queue receives a new frame, it is added back to the heap and the heap is sorted. Therefore, each frame requires a few operations to calculate its time stamp and $O(\log_2 n)$ operations (for n services) to sort the heap once or twice. A simple method to set the service rates is to measure the traffic allocated to each service by the classifier over a period of time, dynamically calculate the f_i fractions and use them as service rates. If a higher layer scheduler is used, the frames will already be scheduled as desired. Therefore, this scheme transparently preserves higher layer scheduling decisions.

III. Simulation Setup

In order to evaluate our architecture we performed extensive tests using the ns-2 simulator [10], extended with additional wireless error modules, link layer schemes and application models. We repeated each experiment 30 times, using the random seeds embedded in ns-2. The results shown below represent averages from all runs. *HSCSD* links simulate the *High Speed Circuit Switched Data* service of GSM, which bundles multiple circuit switched links to increase bandwidth. We used a bandwidth of 86.4 Kbps with 100 byte frames. To randomize losses, bit interleaving is used, simulated by a 100 ms delay, so we used independent frame losses at rates of 1%, 2%, 5% and 10%.

WLAN links simulate an IEEE 802.11b system with 5 Mbps of bandwidth and a 3 ms delay, using 1000 byte frames. WLAN links are treated as full-duplex for simplicity. To allow comparisons with other studies, WLAN links corrupt bits at exponentially distributed intervals with average durations of 2^{14}, 2^{15}, 2^{16} or 2^{17} bits [6], leading to frame loss rates of 0.8%, 1.5%, 3% and 5.9%, respectively. We also ran experiments under error free conditions for reference. We ignored higher layer headers as they uniformly influence all schemes, but accounted for the *exact* framing overhead required by each link layer scheme.

We simulated two topologies, shown in Fig. 3. In the two wireless link topology (solid frame) wireless host A (WH A) communicates with wireless host B (WH B), simulating peer-to-peer communication. Both wireless links are of the same type. In the one wireless link topology (dotted frame), base station A (BS A) communicates with WH B, via base station B (BS B). Most data flows from BS A to WH B, simulating

client-server communication. For WLAN tests the wired link was a 10 Mbps LAN with 1 ms delay. For HSCSD tests it was a 2 Mbps WAN with 50 ms delay.

We simulated two TCP applications and one UDP application in parallel operation, with different link layer schemes for each application class. For TCP, we simulated file transfer and World Wide Web (WWW) browsing, over TCP Reno with 500 ms timers. For UDP, we simulated continuous media distribution, a delay sensitive error tolerant application. All applications used the same path, with the TCP servers and UDP sender at one end and the TCP clients and UDP receiver at the other. All applications started together and the test ended when the file transfer ended.

We used the link layer schemes that were found to be most effective for TCP and UDP in single application experiments [7], offering one TCP and one UDP service. The classifier assigned packets to services based on their IP DS fields [8] which were set by the applications. File transfer and WWW browsing traffic used the *same* service. The rate table for the scheduler was set statically so that the UDP application was guaranteed its *peak* bandwidth, *before* adding link layer overhead. Since continuous media distribution was not constantly active, the average bandwidth available for TCP was higher than implied by its rate.

We first simulated a *Raw Link* scheme, which does not perform any error recovery, but with distinct services for TCP and UDP. The other experiments combine three TCP oriented schemes with one UDP oriented scheme. For TCP we tested reliable schemes that deliver frames in sequence to avoid triggering TCP retransmissions. In *Selective Repeat* the sender buffers outgoing frames and retransmits unacknowledged frames after a timeout. The receiver acknowledges frames received in sequence, buffers out of sequence frames and returns *negative* ACKs (NACKs) for each gap, allowing the sender to retransmit only lost frames. The variant used allows multiple NACKs per loss [11]. Each frame includes sequence and ACK numbers (2 bytes).

To avoid conflicts with TCP retransmissions, the sender in *Karn's RLP* abandons frames after 3 retransmissions [4]. Thus, the sender never stalls, indefinitely retransmitting the same frame. In this scheme only NACK and *keepalive* messages during idle periods are needed, thus frames only include a sequence number (1 byte). *Berkeley Snoop* is a TCP aware scheme [6]. The wireless base station *snoops* inside TCP segments, buffering data sent to the wireless host. If duplicate TCP ACKs indicate a loss, the packet is retransmitted by the base station and the ACKs are suppressed. Snoop does not incur any link layer header overhead.

Continuous media distribution prefers low delay over full reliability. While in sequence delivery is critical for TCP, it is useless for this playback application. We thus tested this application over our *Out of Sequence* (OOS) variant of Karn's RLP, which immediately releases all received frames to higher layers, with up to 1 retransmission per loss [7]. With OOS RLP, when a frame is lost subsequently received frames are not delayed waiting for the lost one.

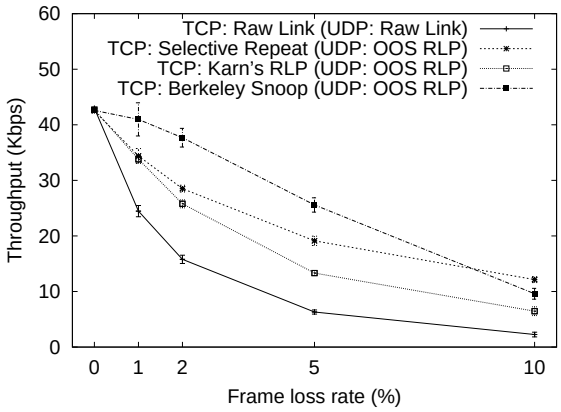

Fig. 4. FTP throughput, one HSCSD link

IV. File Transfer

The first TCP application simulated was file transfer over FTP. We sent a large file from a wireless or wired server to a wireless client (see Section III). The file sizes were 10 MBytes for HSCSD and 100 MBytes for WLAN experiments, large enough to provide stable results but small enough to complete within a reasonable period. When the file transfer completed, the simulation ended for *all* applications. FTP sends data as fast as possible, with TCP in complete control of flow and congestion control. We measured application throughput, defined as the amount of *application* data transferred divided by total time. Retransmissions are *excluded* from this metric as they signify overhead.

Fig. 4 presents file transfer throughput in the one HSCSD link scenario, showing for each curve the scheme used for TCP (in parentheses, the scheme used for UDP). The throughput curves are similar to those of single application experiments, but with lower average throughput due to contention with the other applications. All schemes provide considerable improvements over Raw Link, with Karn's RLP lagging behind Selective Repeat. Berkeley Snoop performs best, except at the highest loss rate, due to its lower overhead, an important factor for low bandwidth links. Its performance improvement relative to the other schemes with increasing loss rates is due to a corresponding performance drop with WWW browsing (see Section V).

Fig. 5 shows the corresponding results for the two WLAN link scenario, where Selective Repeat is again ahead of Karn's RLP. The crucial change from the previous case is that Berkeley Snoop performs only slightly better than Raw Link. This phenomenon, also evident in single application tests, appears in this topology because Berkeley Snoop only performs retransmissions from the base station. As a result, local recovery is performed over one wireless link in the path only, with losses over the other one handled by TCP.

Overall, our file transfer throughput results are similar to those of single application tests. Selective Repeat and Karn's RLP provide considerable gains regardless of the underlying wireless link and topology. Berkeley Snoop works well only in single wireless link topologies, with data transfers from the wired towards the wireless host. The changes from single ap-

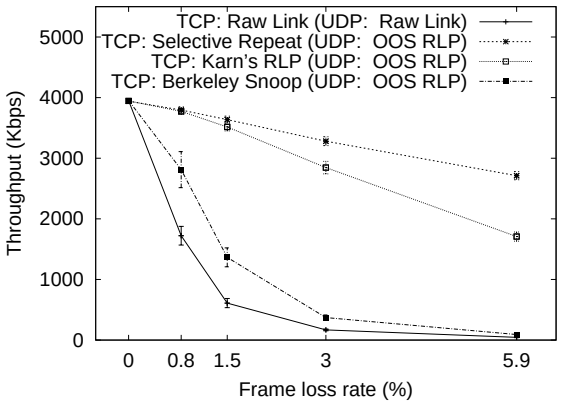

Fig. 5. FTP throughput, two WLAN links

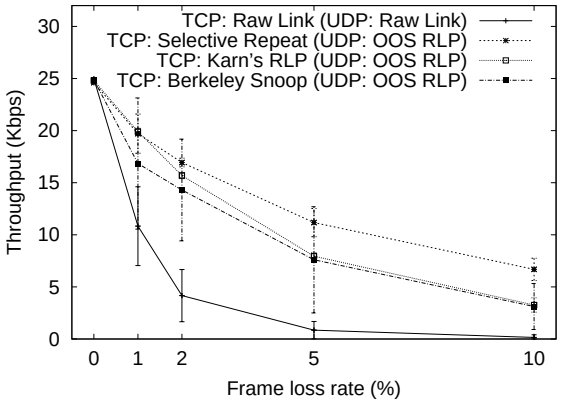

Fig. 6. HTTP throughput, one HSCSD link

plication tests are the lower throughput and the performance gains of Berkeley Snoop with higher losses due to lower WWW browsing performance.

V. World Wide Web Browsing

The second TCP application simulated was World Wide Web (WWW) browsing over HTTP. A WWW client accesses *pages* containing text, links and embedded objects, from a WWW server. The interaction consists of transactions: the client requests a page, the server returns it, the client requests each embedded object, and the server returns them. The ns-2 HTTP module uses empirical distributions to draw request, page and embedded object sizes and the number of embedded objects per page [12].

WWW browsing was simulated between a wired or wireless server and a wireless client (see Section III) until the simultaneous file transfer completed. We measured WWW browsing throughput, defined as the amount of *application* data transferred from the server to the client divided by total time. Client requests are reflected in WWW browsing throughput by introducing delays between data transfers. The HTTP module of ns-2 provided measurements only at the end of each transaction, thus the performance metrics for *all* applications were calculated at the end of the last completed WWW transaction within the simulation period. Only one transaction was in progress at any time.

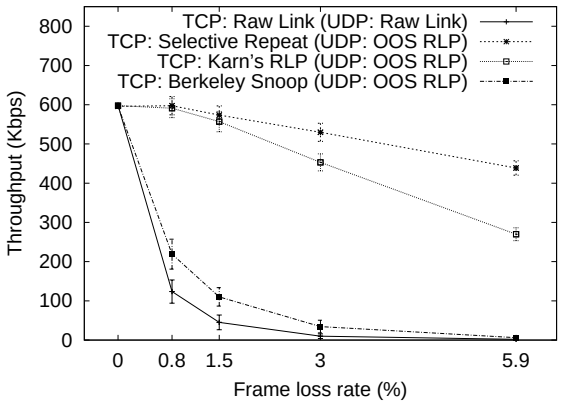

Fig. 7. HTTP throughput, two WLAN links

Fig. 6 presents WWW browsing throughput in the one HSCSD link scenario. Selective Repeat provides the best performance, followed by Karn's RLP. In contrast to FTP tests, Berkeley Snoop lags behind both due to its topological limitations, even though its lower overhead is critical for low bandwidth links. While most data flows in the wired to wireless direction, client requests in the reverse direction are critical for performance. Berkeley Snoop cannot retransmit both client and server data, thus it cannot optimize WWW browsing performance. The resulting drop in WWW browsing throughput explains why file transfer throughput increased with higher loss rates in this scenario.

Fig 7 shows WWW browsing throughput in the two WLAN link scenario. The relative performance of most schemes is very similar to the previous case, with Selective Repeat ahead of Karn's RLP. However, Berkeley Snoop is only marginally better than Raw Link, since it cannot retransmit client data over one wireless link and server data over the other, thus resorting to TCP in both directions. The relative performance of all schemes is quite similar to the corresponding file transfer throughput results.

Overall, our WWW browsing throughput results are similar to those of single application tests. Both TCP unaware schemes offer considerable throughput improvements regardless of the underlying wireless link and topology. The short bidirectional transfers of WWW browsing differentiate it from the unidirectional file transfers, showing that FTP is unable to capture the behavior of interactive applications. This is clearly demonstrated by the performance problems of Berkeley Snoop with WWW browsing.

VI. Continuous Media Distribution

The UDP application simulated was continuous media distribution, where a single speaker sends audio, and possibly video, to a wireless client. The speaker alternates between *talking* and *silent* states with exponential durations, averaging 1 s and 1.35 s, respectively. When talking, the speaker transmits data at a *Constant Bit Rate* (CBR) of 14.4 Kbps for HSCSD and 1 Mbps for WLAN. The rate tables in each multi-service link

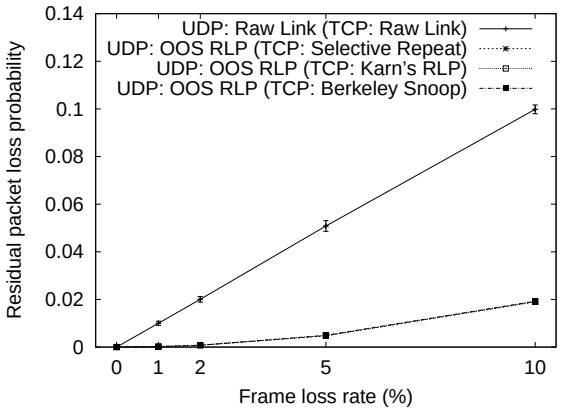

Fig. 8. CBR loss, one HSCSD link

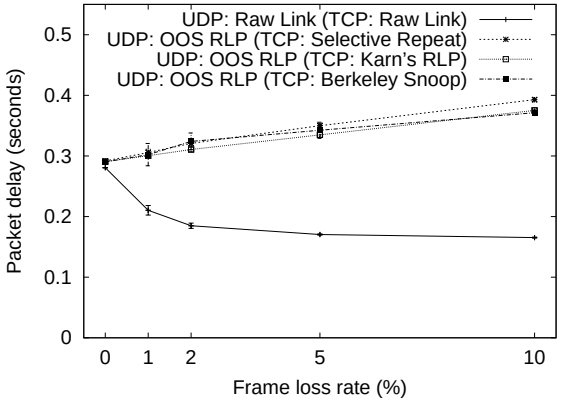

Fig. 9. CBR delay, one HSCSD link

guaranteed this bandwidth to the UDP service, to reduce contention with the TCP service. Since the UDP application is active only 42.5% of the time, the average bandwidth available for TCP was 80.3 Kbps in HSCSD tests and 4.575 Mbps in WLAN tests.

We assumed that the UDP application used an error recovery mechanism to tolerate some congestion induced losses and that received packets were buffered up to a *playback point* for smooth playback. To characterize application performance we measured the *residual* loss rate at the receiver after link layer recovery. Since packets missing the playback point are dropped, we also calculated a delay metric covering *most* packets. We used mean packet delay plus twice its standard deviation to account for delay variance. Fig. 8 shows residual loss in the one HSCSD link scenario. Each curve indicates the scheme used for UDP (in parentheses, the scheme used for TCP). Residual loss was exactly the same as in single application experiments, since the schemes used operate in the exact same manner.

Delay for the one HSCSD link scenario is shown in Fig. 9. Since the link only transmits a single frame at any time, with any non-preemptive scheduler contention between TCP and UDP will increase delay for both. The scheduler allocates sufficient bandwidth for the UDP application though, thus it suffers only a modest delay increase. The differences between the OOS RLP curves are explained by the aggregate TCP performance provided by the corresponding TCP scheme. With Raw Link delay *drops* with increasing loss rates, as TCP throughput drops dramatically. Delay in the two HSCSD link scenario, shown in Fig. 10, is similar to that in the previous case. The exception is Berkeley Snoop, since in this scenario its performance for both TCP applications lags behind Selective Repeat and Karn's RLP, leading to the smallest delay increase for UDP.

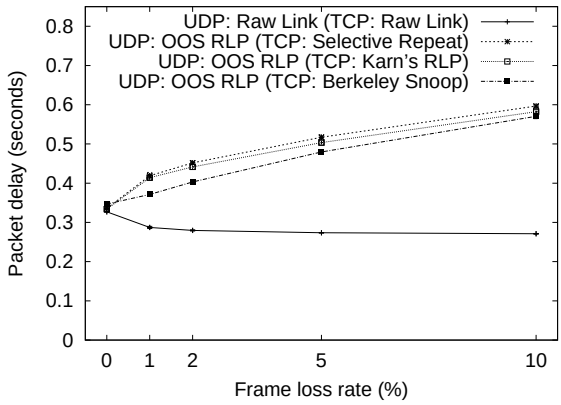

Fig. 10. CBR delay, two HSCSD links

Overall, our continuous media distribution results indicate that while loss rates are identical to single application experiments, the contention between TCP and UDP leads to a delay increase for UDP traffic. Since part of the delay increase is due to the retransmissions of OOS RLP, it seems that the scheduler manages to maintain delay at acceptable levels, without preventing the TCP applications from exploiting as much bandwidth as possible.

VII. Conclusions

We presented a multi-service link layer architecture that attempts to enhance the performance of diverse applications by supporting multiple error recovery schemes in parallel. Our results extend our previous work [7] by testing WWW browsing along with file transfer and continuous media distribution. Based on these results we can draw some conclusions regarding the multi-service link layer architecture.

First, application performance was similar to that of single application tests with the same link layer scheme, thus the architecture provides the performance gains of each individual scheme. Second, the TCP unaware schemes improved both file transfer and WWW browsing performance, thus there is no need to provide separate services for different TCP applications. Third, the additional UDP delay due to contention with TCP was kept low by the scheduler, thus the scheduler effectively protects services from each other. Fourth, these gains were achieved by the exact same schemes as in single application tests, thus existing code can be used as is. Fifth, the best performing schemes were transport layer unaware, thus our architecture can provide enhanced performance without violating protocol layering.

REFERENCES

[1] G. Xylomenos and G. C. Polyzos, "Internet protocol performance over networks with wireless links," *IEEE Network*, vol. 13, no. 5, pp. 55–63, July/August 1999.

[2] B. R. Badrinath, A. Bakre, T. Imielinski, and R. Marantz, "Handling mobile clients: A case for indirect interaction," in *Proc. of the 4th Workshop on Workstation Operating Systems*, 1993, pp. 91–97.

[3] T. Goff, J. Moronski, D.S. Phatak, and V. Gupta, "Freeze-TCP: A true end-to-end TCP enhancement mechanism for mobile environments," in *Proc. of the IEEE INFOCOM '00*, 2000, pp. 1537–1545.

[4] P. Karn, "The Qualcomm CDMA digital cellular system," in *Proc. of the USENIX Mobile and Location-Independent Computing Symposium*, 1993, pp. 35–39.

[5] A. DeSimone, M. C. Chuah, and O.C. Yue, "Throughput performance of transport-layer protocols over wireless LANs," in *Proc. of the IEEE GLOBECOM '93*, 1993, pp. 542–549.

[6] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," in *Proc. of the ACM SIGCOMM '96*, 1996, pp. 256–267.

[7] George Xylomenos and George C. Polyzos, "Quality of service support over multi-service wireless Internet links," *Computer Networks*, vol. 37, no. 5, pp. 601–615, 2001.

[8] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for Differentiated Services," RFC 2475, December 1998.

[9] S. Golestani, "A self-clocked fair queueing scheme for broadband applications," in *Proc. of the IEEE INFOCOM '94*, 1994, pp. 636–646.

[10] "UCB/LBNL/VINT Network Simulator - ns (version 2)," Available at http://www.isi.edu/nsnam.

[11] P. T. Brady, "Evaluation of multireject, selective reject, and other protocol enhancements," *IEEE Transactions on Communications*, vol. 35, no. 6, pp. 659–666, June 1987.

[12] B. A. Mah, "An empirical model of HTTP network traffic," in *Proc. of the IEEE INFOCOM '97*, 1997, pp. 592–600.