Multi-Service Wireless Internet Link Enhancements

George Xylomenos,
Department of Informatics, Athens University of Economics and Business, Greece,
xgeorge@aueb.gr
George C. Polyzos,
Department of Informatics, Athens University of Economics and Business, Greece,
polyzos@aueb.gr

Abstract

The deployment of several real-time multimedia applications over the Internet has motivated a considerable research effort on the provision of multiple services over the Internet. In order to extend this work over wireless links however we must also take into account the performance limitations of the medium. We survey various related approaches and conclude that link layer schemes provide a universal and localized solution. Based on simulations of application performance over various link layer schemes we show that different approaches work best for different applications. We present a multi-service link layer architecture which enhances the performance of diverse applications by concurrently supporting multiple link layer schemes. Simulations of multiple applications executing simultaneously show that this approach dramatically improves performance for all. We finally present various ways of embedding this approach into the Internet.

Index Terms

Wireless link layer protocols, quality of service, differentiated services.

I. Introduction

The most important Internet related developments in the past few years were arguably the emergence of mechanisms for providing *Quality of Service* (QoS) guarantees to applications and the wide deployment of wireless access networks using technologies such as *Digital Cellular Communications* and *Wireless Local Area Networks* (WLANs). The QoS effort is largely driven by the desire to migrate applications with real-time delay constraints, such as voice telephony and video conferencing, from circuit-switched networks to the packet-switched Internet. Multiple approaches have been proposed for the provision of adequate QoS for such applications, without dramatic changes to the Internet [1], [2]. These schemes provide differentiated treatment, in terms of priority and drop probability, to different traffic classes, so as to combine the economies of statistical multiplexing with the delay guarantees needed for real-time applications.

Integrating wireless networks into the Internet is relatively simple, due to the physical layer independence of the *Internet Protocol* (IP), which offers a standard interface to higher layers. However, while providing IP services over wireless links is easy, the resulting performance is often disappointing due to the relatively high error rates of wireless links. In addition, while microwave and satellite links have long been part of the Internet, higher layer protocols commonly make assumptions about link performance that cannot be met by wireless links, leading to further performance degradation.

This chapter describes an architecture that improves the performance of diverse Internet applications while extending Internet QoS support over wireless networks. In Sec. II we outline the problem and review previous work on wireless link enhancements and service differentiation. In Sec. III we describe the simulation setup that was used throughout the chapter. In Sec. IV we present single-service simulations showing that different applications favor different link enhancement schemes. In Sec. V we present a *multi-service link layer* architecture that simultaneously enhances the performance of diverse applications by supporting multiple link mechanisms in parallel. In Sec. VI we present simulations showing that with this architecture each application achieves similar gains as when it operates by itself over its preferred link layer mechanism. We then discuss the integration of this approach with various Internet QoS schemes: Sec. VII covers the existing best-effort service, Sec. VIII the Differentiated Services architecture, and Sec. IX a dynamic service discovery architecture.

II. Background and Related Work

IP offers an unreliable packet delivery service between any two Internet hosts. IP packets may be lost, reordered or duplicated. Applications can use the *User Datagram protocol* (UDP) for direct access to this service. Some applications employ UDP assuming that the network is reliable enough for their needs, for example, file sharing via NFS over wired LANs. Delay sensitive applications may also use UDP in order to employ customized loss recovery mechanisms. For example, delay sensitive applications may add redundancy to their data to tolerate some losses without (slow) end-to-end retransmissions.

Most applications however require complete reliability, hence they employ the *Transmission Control Protocol* (TCP), which offers a reliable byte stream service. TCP breaks the application data stream into segments which are reassembled at the receiver. The receiver returns cumulative *acknowledgments* (ACKs) for segments received in sequence, with duplicate ACKs for out of sequence ones. Since IP may reorder packets, the sender retransmits the first unacknowledged segment only after receiving multiple (usually 3) duplicate ACKs. The sender tracks the round trip delay of the connection, so that if an ACK for a segment does not arrive in time, the segment is retransmitted [3]. Due to the high reliability of wired links, TCP assumes that all losses signify congestion. Therefore, after a loss is detected, TCP reduces its transmission rate, and then increases it slowly so as to gently probe the network [3].

With wireless links, some common assumptions about network reliability are no longer valid. The high error rate of wireless links causes UDP application performance to degrade or even become unacceptable. On the other hand, TCP continuously reduces its transmission rate due to wireless errors to avoid what it (falsely) assumes to be congestion. With multiple wireless links on the path, throughput is reduced in a multiplicative manner [4]. WLANs depict losses of up to 1.5% for Ethernet size frames [5], while cellular systems suffer from losses of 1–2% for their much shorter frames [6]. The effects of these losses are dramatic: a 2% frame loss rate over a WLAN halves TCP throughput [7].

The performance of UDP applications over wireless links has not been studied extensively, not only due to their diversity, but also because they were perceived as oriented to wired LANs, an assumption challenged by media streaming over the Internet. Considerable work has been devoted to TCP however. Most TCP enhancements try to avoid triggering congestion recovery due to wireless errors. Generic TCP enhancements such as *Eifel* [8] and *Selective Acknowledgments* [9] improve TCP performance by reducing the *number* of redundant TCP retransmissions, without however reducing their (end-to-end) *delay*.

A wireless link oriented approach is to *split* TCP connections into one connection over the wireless link and another one over the wired part of the path, bridged by an agent [10]. Recovery is only performed locally over the wireless link. This scheme violates the end-to-end semantics of TCP and it is incompatible with IP security which encrypts TCP headers [11]. Other approaches maintain TCP semantics by *freezing* TCP state whenever persistent errors are detected [12], [13]. As long as errors persist, TCP does not invoke congestion control. Wireless errors and congestion losses are differentiated using either explicit loss notifications [12] or explicit congestion notifications [13]. Thus, performance is not unnecessarily degraded, but error recovery remains end-to-end and the network must support explicit loss or congestion notifications.

The main alternative to TCP modifications is local error recovery at the link layer. One option is to use the *Radio Link Protocols* (RLPs) of cellular systems which provide error control customized for the underlying link [6], [14]. Since they only apply a single error control mechanism though, RLPs may be inappropriate for some traffic. For example, retransmissions are beneficial to TCP but delay real-time traffic. Another problem is that RLPs may interfere with TCP recovery [15], leading to conflicting retransmissions between the link and transport layers. This is avoided by only performing retransmissions when the link layer recovers from losses much faster than the transport layer [16].

One method of jointly optimizing recovery between layers is to exploit transport layer information at the link layer. By *snooping* inside *each* TCP stream at the wireless base station, we can transparently retransmit lost segments when duplicate ACKs arrive, hiding the duplicates from the sender to avoid end-to-end recovery. This approach avoids both control overhead and adverse cross layer interactions [7]. However, it is incompatible with IP security as it employs TCP header information and it only works in the direction from the wired Internet towards the wireless host due to its reliance on TCP ACKs. In the reverse direction, ACKs are returned late and may signify congestion losses [4].

At the transport layer, each application may select a protocol suitable for its needs, but at the link layer all traffic uses the same protocol. Link layer protocols usually offer a single service, which may not be able to satisfy the requirements of all types of traffic. It has been suggested that link layer traffic should be split in two classes, TCP and UDP-based, so as to provide reliable transmission for TCP traffic only. Such a link layer may be either aware or unaware [17], [18] of TCP semantics, as long as it can distinguish TCP from UDP packets. This method however does not improve the performance of UDP applications, which may then fail over the error prone wireless links.

In addition to providing differentiated error recovery at the link layer, it is also possible to provide different service priorities over wireless links, for example, by appropriately modifying the contention interval for each traffic class in a WLAN [19], [20]. This allows the QoS mechanisms of the wired Internet to be directly extended over wireless links. Since these mechanisms are not particular to wireless links however, they may be considered as orthogonal to the provision of customized error control for each type of traffic.

Link layer approaches have the advantage over higher layer ones of providing a local solution that does not require any changes to higher layers. This makes them transparent to the rest of the Internet, enables them to recover faster than transport layer solutions and allows them to exploit lower layer information to optimize recovery [16], [18]. Therefore, for the remainder of this chapter we will focus on link layer mechanisms providing customized error control to each traffic class. Then, we will examine the integration of such mechanisms with various approaches for QoS provision over the Internet.

III. Simulation Setup

To study the interactions between link layer schemes, transport protocols and applications, we performed extensive simulations using the ns-2 simulator [21], enhanced with additional error modules, link layer schemes and applications [22]. To compensate

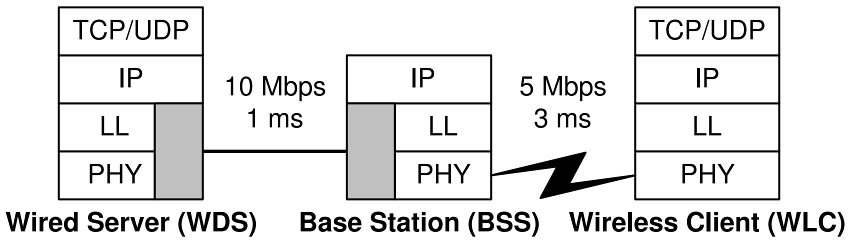

Fig. 1. Simulation topology.

for statistical fluctuations, we repeated each experiment 30 times, using the random seeds embedded in ns-2. The results shown below represent averaged values from all runs, with error bars at plus/minus one standard deviation. We provide elsewhere additional simulation details, as well as results for other network topologies and wireless links [23], [24]. We only provide here a sample of our results in order to motivate our architecture.

The simulated network topology is shown in Fig. 1. A *Wired Server* (WDS) communicates with a *Wireless Client* (WLC) via a *Base Station* (BSS). The wired link between the WDS and the BSS is a LAN with 10 Mbps of bandwidth and a 1 ms delay. The wireless link between the BSS and the WLC simulates an IEEE 802.11b WLAN with 5 Mbps of bandwidth and a 3 ms delay, using 1000 byte frames. All links are treated as full-duplex pipes for simplicity. To allow comparisons with other studies, the wireless link corrupts bits at exponentially distributed intervals with average durations of 2^{14}, 2^{15}, 2^{16} and 2^{17} bits [7], leading to frame loss rates of 0.8% to 5.9%. We also ran experiments under error-free conditions for reference purposes. The error processes in each wireless link direction are identical but independent. We ignored TCP/UDP/IP headers as they uniformly influence all link layer schemes, but accounted for the *exact* framing overhead required by each link layer scheme.

We tested both TCP and UDP applications, so as to evaluate the suitability of various link layer schemes for diverse applications. For TCP, we simulated large file transfers and World Wide Web browsing, using the TCP Reno module of ns-2 with 500 ms granularity timers. For UDP, we simulated continuous media distribution, a delay sensitive but error tolerant application. In all cases, most data flows from the WDS to the WLC, simulating a client-server interaction with the client on a wireless network. Some data also flow in the reverse direction, for example, TCP ACKs. As a baseline, we simulated a *Raw Link* scheme, which does not perform any error recovery, under either TCP or UDP.

For TCP applications, we tested reliable link layer schemes delivering frames in sequence to higher layers so as to avoid TCP retransmissions. In *Go Back N* the sender buffers outgoing frames and retransmits unacknowledged frames after a timeout. The receiver positively acknowledges frames received in sequence, dropping out of sequence ones. After a timeout the sender retransmits *all* outstanding frames. *Selective Repeat* improves upon this by buffering out of sequence frames at the receiver and returning *negative* ACKs (NACKs) when it detects gaps, thus allowing the sender to retransmit lost frames only. Our Selective Repeat variant allows multiple NACKs per loss [25]. In both schemes, under persistent losses the sender may exhaust its window and stall. Each frame includes sequence and acknowledgment numbers (2 bytes).

To prevent conflicts with TCP retransmissions due to multiple link layer retransmissions [15], the sender in *Karn's RLP* abandons frames not received after 3 retransmissions (for TCP) [6]. Thus, the sender never stalls. This scheme uses only NACK and *keepalive* frames during idle periods, thus frames only include a sequence number (1 byte). *Berkeley Snoop* is a TCP aware scheme [7], included in the ns-2 distribution. A module at the BSS *snoops* inside TCP segments, buffering data sent to the WLC. If duplicate TCP ACKs indicate a lost packet, this is retransmitted by the BSS and the ACKs are suppressed.

For the UDP-based continuous media application, low delay is preferable to full reliability. *Forward Error Correction* (FEC) schemes offer limited recovery by adding redundancy to the transmitted stream, allowing the receiver to recover from losses. The *XOR based FEC* scheme sends data frames unmodified, but every 12 frames (a tradeoff between overhead and delay) a *parity* frame is also transmitted, generated by XOR'ing the preceding data frames, collectively called a *block*. If a *single* data frame is lost from a block, we can recover it by XOR'ing the remaining data frames with the parity frame. A timeout is used to prematurely emit a parity frame when the link becomes idle before the current block has been completed. All frames include a sequence number (1 byte).

The UDP-based application was also tested with Selective Repeat, in order to examine the interactions between a fully reliable scheme and a delay sensitive application. We also tested Karn's RLP, but with 1 retransmission per loss to keep delay low. In this scheme, frame losses cause subsequently received frames to wait until the missing one is received or abandoned. This is detrimental to applications with their own resequencing (playback) buffers, a common case in continuous media distribution. Our *Out of Sequence* (OOS) variant of Karn's RLP immediately releases all received frames to higher layers so as to avoid such delays. This variant was also tested with 1 retransmission per loss.

IV. Single-Service Link Layer Performance

The first TCP-based application tested was file transfer over FTP. We simulated a 100 MByte file transfer from the WDS to the WLC. File transfers are unidirectional, with only TCP ACKs moving in the reverse direction. The ns-2 FTP module sends data as

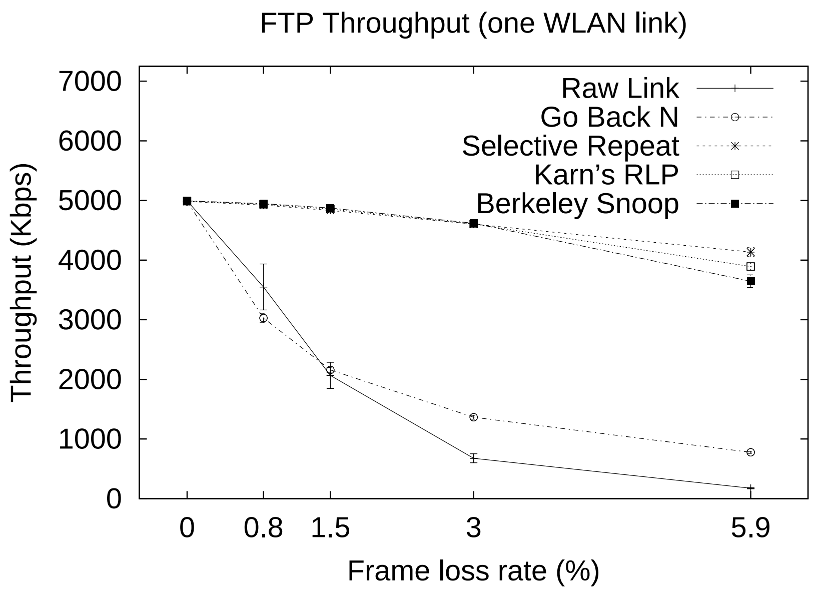

Fig. 2. Stand-alone file transfer: throughput.

fast as possible, with TCP handling flow and congestion control. As TCP completely controls FTP behavior, performance studies usually rely on FTP transfers in the wired to wireless direction, assumed to be the most common case, in order to characterize overall TCP performance [7]. We measured application throughput, i.e. the amount of *application* data transferred divided by total time. TCP and link layer retransmissions are *not* included as they signify overhead from an application's viewpoint.

Fig. 2 shows the FTP throughput achieved by each link layer scheme for a range of error rates, averaged over 30 experiments. Most schemes offer large, but similar, performance gains over Raw Link. The exception is Go Back N, due to its naive strategy of retransmitting all outstanding frames after a loss. Due to the high speed of the wireless link, at any given time there are multiple frames in flight over the link. Therefore, retransmitting all outstanding frames after each loss wastes a lot of bandwidth. The other three enhancement schemes are only differentiated at high loss rates, where the most persistent scheme, Selective Repeat, is ahead of Karn's RLP and Berkeley Snoop.

Most studies of wireless TCP performance use FTP since it is easy to simulate and it directly reflects TCP behavior. Real applications however make many *short* data exchanges, thus TCP rarely reaches its peak throughput. In addition, most applications are either interactive or employ request/reply protocols, thus data flows in *both* directions, and *each* exchange must complete for the application to proceed. Therefore, we also simulated *World Wide Web* (WWW) browsing over HTTP [26], the most popular Internet application.

A WWW client accesses *pages* containing text, links and embedded objects, stored on a WWW server. The client-server interaction consists of *transactions*: the client requests a page from a server, the server returns the page, the client requests each embedded object, and the server returns them. All transfers are performed over TCP. The ns-2 HTTP module provides empirical distributions for the request, page and embedded object sizes, as well as the number of objects per page [26]. Only one transaction is in progress at any time and there are no pauses between transactions. WWW browsing was simulated between the WDS and the WLC for 500 s. We measured WWW browsing throughput, i.e. the amount of *application* data transferred from the WDS to the WLC, including both pages and embedded objects, divided by total time.

Fig. 3 shows the WWW browsing throughput achieved by each link layer scheme. Since the short data transfers of WWW browsing rarely reach high speeds, the naive retransmission strategy of Go Back N does not cause a very dramatic performance deterioration. Karn's RLP performs slightly better than Selective Repeat again due to the small transfers, since its keepalive feature allows it to recover fast from losses when the link becomes idle, unlike Selective Repeat which has to wait for a timeout to detect losses. The major difference is in the performance of Berkeley Snoop, which provides only minor improvements over Raw Link, as it does not retransmit in the WLC to BSS direction. This shows that even if most traffic is in the server to client direction, in an interactive application the client to server traffic is equally critical for overall performance.

While applications using UDP due to its simplicity would work well with these schemes, delay sensitive applications would face serious problems with a fully reliable retransmission scheme. We thus tested UDP performance using real-time continuous media distribution, i.e. a lecture where a speaker at the WDS sends audio and video to an audience including the WLC. We used a speech model where the speaker alternates between *talking* and *silent* states with exponential durations, averaging 1 s and 1.35 s, respectively [27]. Media are only transmitted in the talking state. When talking, the speaker transmits packets isochronously at a *Constant Bit Rate* (CBR) of 1 Mbps. We simulated continuous media distribution for 500 s. We assumed that the application

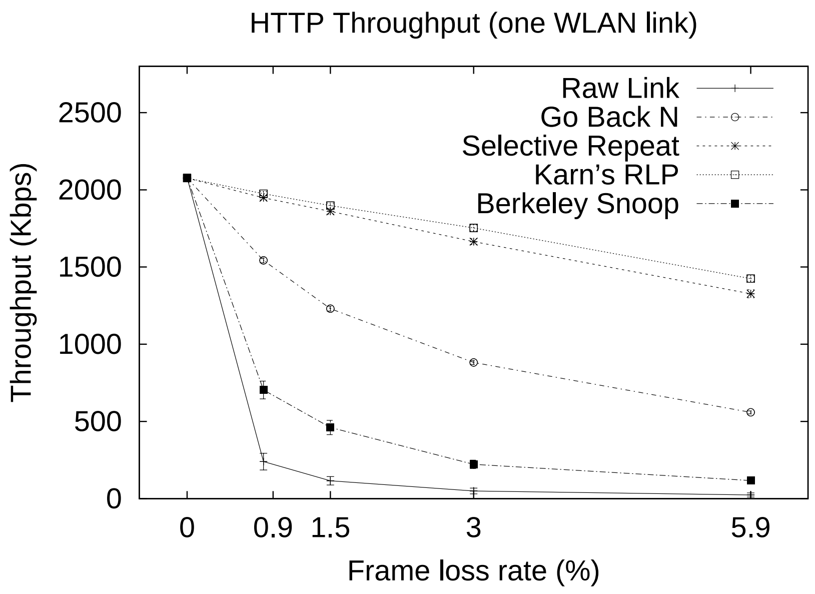

Fig. 3. Stand-alone WWW browsing: throughput.

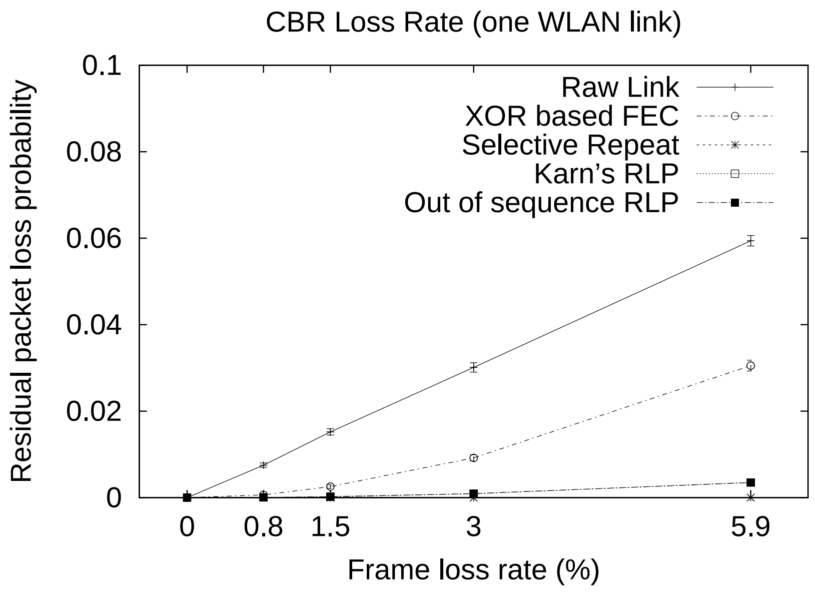

Fig. 4. Stand-alone continuous media distribution: loss.

uses FEC to tolerate some loss without retransmissions and that received packets are buffered until a *playback point* determined by human perception. To characterize performance, we measured the *residual* loss rate at the receiver after link layer recovery. Since packets missing the playback point are dropped, loss rate is coupled with a delay metric covering *most* packets. We used mean packet delay plus twice its standard deviation, so as to account for variable delays.

Fig. 4 shows residual loss with each scheme. Both RLP schemes dramatically reduce losses, even with a retransmission limit of 1. XOR based FEC depicts gains that do not justify its overhead. For example, by introducing 8.3% of overhead, a native loss rate of 3% is reduced to 1%. Selective Repeat offers full reliability, but at the cost of very high delays. Fig. 5 shows the delay metrics of each scheme with continuous media distribution. The Raw Link curve is flat since no recovery takes place. Selective Repeat exhibits the highest delay due to its full recovery. While both RLP variants perform similarly at low error rates, as link conditions deteriorate only OOS RLP manages to keep delay low, since in sequence delivery causes many packets to be delayed after each loss. Interestingly, XOR based FEC is slower than OOS RLP. While OOS RLP requires a NAK and a retransmission to recover form a loss, the FEC scheme requires half a block of frames to be received, on average, in order to recover a lost frame. Since this wireless link has low delay, high bandwidth and uses large frames, retransmission turns out to be faster than this type

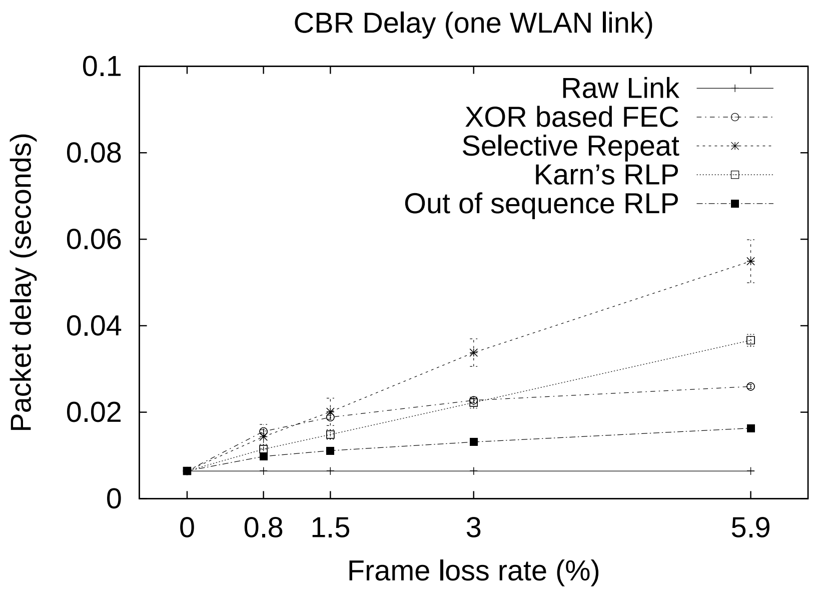

Fig. 5. Stand-alone continuous media distribution: delay.

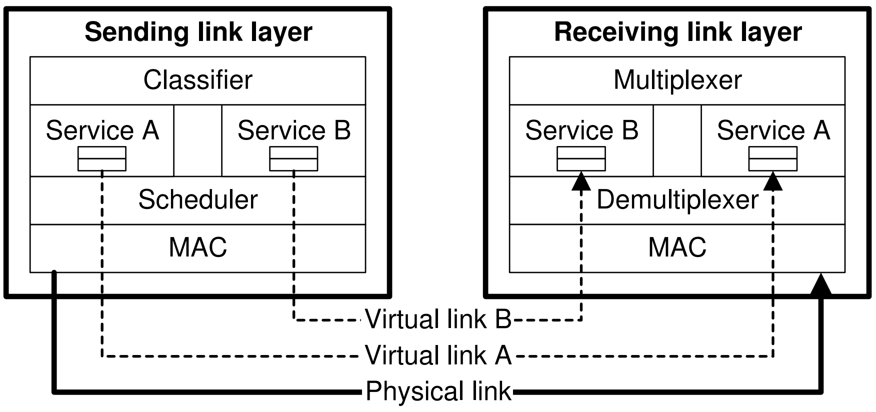

Fig. 6. Multi-service link layer architecture.

of FEC scheme.

V. Multi-Service Link Layer Architecture

The results presented above show that link layer error recovery can considerably enhance Internet application performance over wireless links. They also show however that different schemes work best for the TCP and UDP applications tested. This means that a single link layer scheme, even an adaptive one, *cannot* optimize the performance of all TCP and UDP applications.

We have therefore developed a *multi-service link layer* architecture [28], supporting multiple link enhancement services in parallel over a single physical link. Each service fits the needs of an application class, such as TCP-based or real-time UDP-based. Since the adjacent protocol layers expect the link layer to have single entry and exit points, our architecture provides sublayers to assign incoming packets to services and to multiplex outgoing packets from all services into a single stream. These sublayers keep services unaware of the fact that they are operating within a multi-service context.

Fig. 6 outlines our architecture, showing data flow in one direction. Incoming packets are classified and passed to the most appropriate service, based on their application class. A simple classifier may use the protocol field of the IP header to distinguish between TCP and UDP and the port field of the TCP/UDP headers to determine the application in use. When *Differentiated Services* (DS) are used, the classifier may use the DS field of the IP header [1], which remains visible even with IP security [11]. When *Integrated Services* [2] are used, packets can be classified based on flow state maintained by higher layers. Packets from unknown applications are mapped to the default, best-effort, service.

Each service may employ retransmissions, FEC, or any other mechanism desired, keeping its private buffers, counters and timers. Outgoing frames are passed to a scheduler sublayer which tags each frame with a service number and eventually delivers

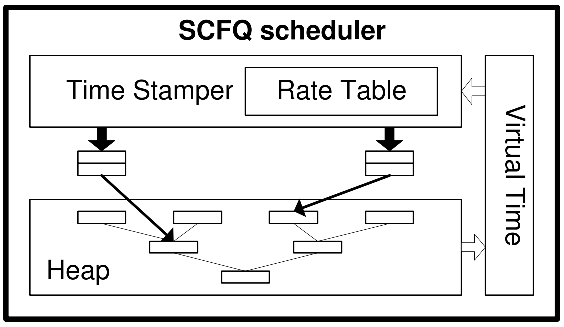

Fig. 7. Self Clocked Fair Queueing frame scheduler.

it to the MAC sublayer for transmission. At the receiver, frames are passed by the MAC sublayer to the demultiplexer sublayer which checks their tags and delivers them to the proper service. Services may eventually release data to the multiplexer sublayer, which passes them to the network layer. Therefore, the peer link layer services communicate over *virtual links*.

As each service may arbitrarily inflate its data stream with error recovery overhead, if we simply transmitted all frames in a FIFO manner we would penalize the services introducing less overhead. Therefore, we have introduced a frame scheduler to ensure that the link is *impartially* shared between services. Impartiality means that each service should receive the same amount of bandwidth as in a single-service link layer. That is, a service performing no error recovery will get the same bandwidth for data in both cases. In contrast, an error recovery service will get the same *total* bandwidth, but it will have to divide it between its data and its error recovery overhead.

We chose a *Self Clocked Fair Queueing* (SCFQ) scheduler [29] which efficiently, but strictly, enforces the desired bandwidth allocations for each service. When some services are idle, their bandwidth is shared among the rest in proportion to their allocations. Fig. 7 gives an outline of the scheduler. The *rate table* holds the fraction of link bandwidth allocated to each service. Frames awaiting transmission are buffered in a separate FIFO queue per service. A virtual time variable is maintained, equal to the *time stamp* of the last packet transmitted. To determine the time stamp of an incoming packet, we divide its size by its service rate, and add the result to the time stamp of the preceding frame in its queue. If its queue is empty, we add the result to the current virtual time. When the link is idle, the frame with the *lowest* virtual time is dequeued, its time stamp becomes the system's virtual time, and the frame is transmitted.

Since the frames entering the FIFO queues have increasing time stamps, we only need to check the head of each queue to determine which frame has the lowest virtual time. The scheduler keeps all non-empty queues in a heap, sorted by the time stamp of their first frame. The heap is sorted when a frame is dequeued for transmission, based on the new head of the corresponding queue. Empty queues are removed from the heap and are re-inserted when they receive a new frame, also causing a heap sort. Thus, each frame requires a simple calculation for its time stamp and $O(\log_2 n)$ operations (for n services) to sort the heap when the frame leaves (and, possibly, when it enters) the scheduler.

This multi-service link layer architecture can be locally deployed over isolated wireless links, transparently to the rest of the Internet. Additional services can be provided by inserting new modules and extending the mappings of the classifier. Services may be optimized for the underlying link, freely selecting the most appropriate mechanisms. The scheduler will ensure that these services will fairly share the link, despite their variable overheads, thus keeping the services unaware of each other.

VI. Multi-Service Link Layer Performance

In order to verify that our architecture can simultaneously enhance the performance of diverse applications, we repeated the simulations of Sec. IV with all applications, i.e. file transfer, WWW browsing and continuous media distribution, executing in parallel but over different link layer services. The TCP servers and UDP sender were at the WDS and the TCP clients and UDP receiver were at the WLC. All applications started together and the simulation ended when the file transfer completed. Our multi-service link layer module for ns-2 provided one TCP and one UDP service, using the link layer schemes that performed best in single application experiments. File transfer and WWW browsing traffic used the *same* link layer service. The classifier assigned packets to services based on their IP DS fields [1] which were set by the applications.

The multi-service link layer module used the scheduler described above. The rate table was set statically so that continuous media distribution was guaranteed its *peak* bandwidth. Although the scheduler allocated 1 Mbps to the UDP service, the average bandwidth available to the TCP service was 4.575 Mbps, since the UDP application was not constantly active. File transfer is generally more aggressive in terms of competing for bandwidth than WWW browsing, as it performs a unidirectional bulk transfer. As this is handled by existing TCP mechanisms however, the link layer did not deal with it.

Fig. 8 presents file transfer throughput results, with each curve showing the service used for TCP (in parentheses, the service used for UDP). These throughput curves are generally similar to those of single application experiments, but with lower average

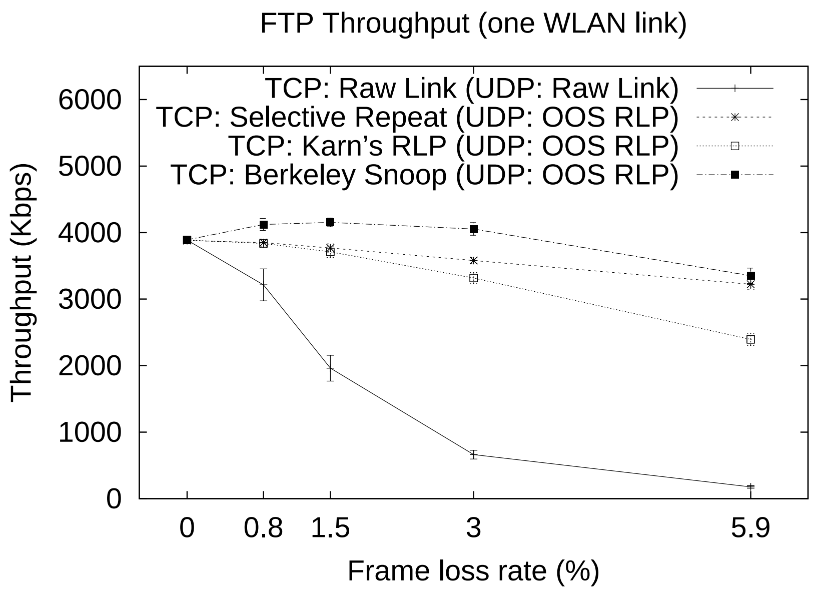

Fig. 8. All applications: file transfer throughput.

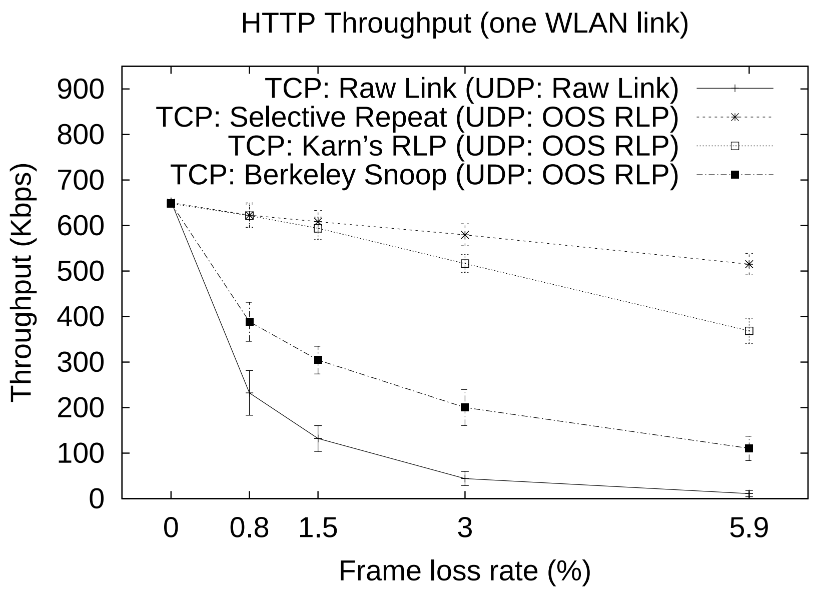

Fig. 9. All applications: WWW browsing throughput.

throughput due to contention with other applications. Selective Repeat performs better than Karn's RLP, especially at higher loss rates, due to its higher persistence. Berkeley Snoop performs best, with its throughput initially increasing with higher loss rates. This is due to a corresponding degradation in WWW browsing throughput with this scheme, as shown below, which leaves more bandwidth available for file transfer.

Results for WWW browsing throughput, shown in Fig 9, are also similar to those of single application experiments. The performance of Selective Repeat and Karn's RLP is quite similar to the file transfer throughput results, with very large performance gains compared to Raw Link. Selective Repeat, due to its persistence, performs better than Karn's RLP as it competes for bandwidth more aggressively. Berkeley Snoop fails to offer significant gains due to its topological limitations, exactly as in single application experiments. This drop in WWW browsing throughput with Berkeley Snoop explains why file transfer throughput initially increased with higher loss rates.

Residual losses for continuous media distribution were the same as in single application experiments, since error recovery is not influenced by contention. The delay metrics, given in Fig. 10, show for each curve the service used for UDP (in parentheses, the service used for TCP). Delay was inflated, an unavoidable effect with our non-preemptive work-conserving scheduler. The

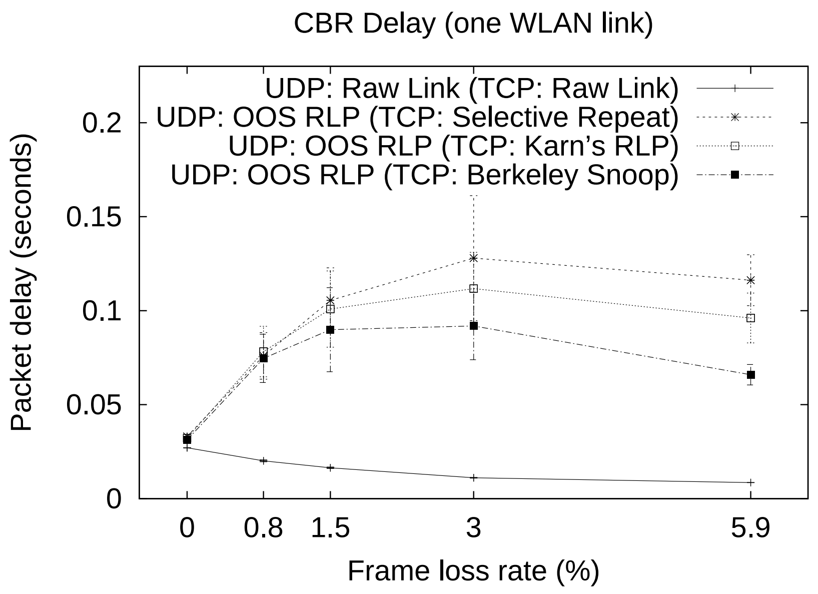

Fig. 10. All applications: continuous media distribution delay.

scheduler managed however to keep delay at reasonable levels, especially considering that the additional delay over Raw Link is partly due to the OOS RLP scheme itself. The reduced delays at higher native loss rates were due to the deteriorating TCP performance which reduced contention. Karn's RLP provided a better balance between TCP throughput and UDP delay than the more persistent Selective Repeat. Berkeley Snoop caused the lowest additional delays, at the cost of reduced aggregate TCP performance.

VII. Best-effort Service Interface

The network layer of the Internet provides a single, best-effort, packet delivery service. Higher layer protocols can extend this service to satisfy additional application requirements. Our simulations show however that multiple link layer services are needed in order to enhance Internet application performance over wireless links. Since higher layer protocols are not aware of multi-service links, they cannot directly map their requirements to available services. To retain compatibility with existing infrastructure, our architecture must perform this mapping transparently. Performance should be at least as good as with a single service, meaning that applications should not be mapped to inappropriate services and that enhancing the performance of one application should not hurt the performance of others. These requirements allow services to be introduced gradually to selectively enhance the performance of some applications.

Since our link layer architecture emulates a single service, the only data that can be used for service selection are the contents of incoming IP packets. We can match application requirements to services using a heuristic classifier which recognizes applications based on IP, TCP and UDP packet header fields. Fig. 11 shows data flow in such a classifier for IPv4 packets. Headers are masked to isolate the fields used for classification. These fields pass through a hashing function that produces an index to a lookup table, whose entries point at the available services. Unrecognized applications are mapped to the default (native) link service which offers the same performance as a single service link layer. The header mask, hashing function and lookup table are provided by an external entity, i.e. an administrator, which is aware of application requirements, header fields and link services.

Higher layers expect the link layer to allocate bandwidth as if a single service was offered, therefore link services should respect this (implicit) allocation. As packets are assigned to services, the classifier measures their size to deduce the share of the link allocated to each service. Over regular time intervals, the classifier divides the amount of data assigned to each service i, by the total amount of data seen, to get a fraction r_i, where $\Sigma_{i=1}^{n} r_i = 1$, for n services. These fractions, or normalized service rates, are entered into the rate table. Applications mapped to the default service are allocated the same bandwidth as with a single service. The other services trade-off throughput for error recovery.

Heuristic packet classification starts with the protocol field of the IP header, indicating TCP, UDP, or another protocol. TCP applications can be all mapped to a single service. For UDP applications however, decisions must be made on a per application basis. Known applications can be detected by examining the source and destination port fields of the UDP header. Another field that may be used is the *Type of Service* (TOS) field of the IPv4 header. Originally, this field was intended to indicate application preferences and packet priorities. This field, however, is rarely used and it has been redefined to support Differentiated Services. This also applies to the *Traffic Class* field of the IPv6 header.

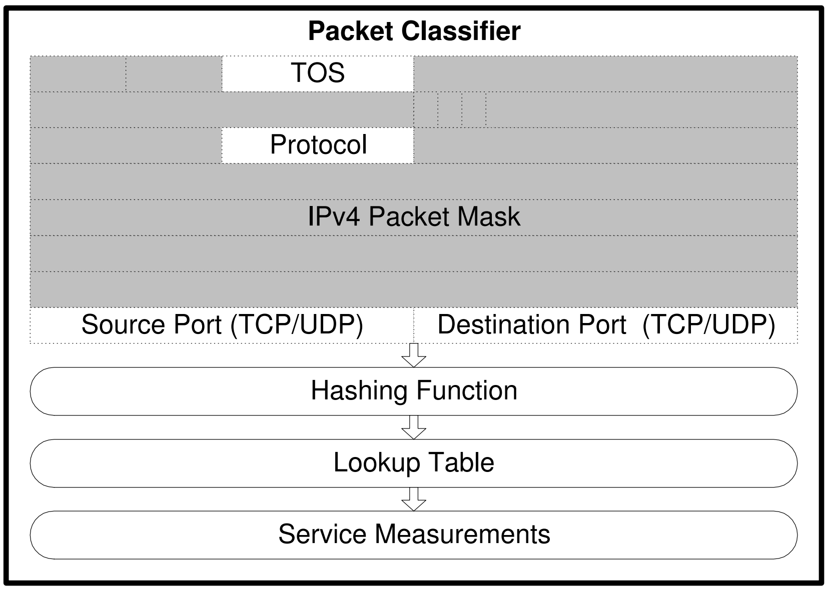

Fig. 11. Heuristic packet classifier.

One drawback of heuristic classifiers is the effort required to construct and maintain them. Applications must be manually matched to services over each type of wireless link whenever new applications or services are added. Many applications will be unrecognized, simply because they do not use fixed ports. Although some higher layer QoS schemes combine the transport protocol and the source/destination port/address fields to classify packets [30], maintaining state for all these combinations requires end-to-end signaling. Another important problem is that IP security mechanisms encrypt the port fields of TCP and UDP headers and put the identifier of the IP security protocol in the protocol field [11], thus hindering heuristic classification.

VIII. Differentiated Services Interface

The best-effort service provided by IP is becoming inadequate as more real-time multimedia applications are deployed over the Internet. To support these applications, some type of performance guarantees must be introduced into the Internet. The main issue for Internet QoS is generally considered to be congestion control, since applications may not be able to get their required throughput due to contention for link resources and their end-to-end delay may increase due to queueing delays. One scheme for Internet QoS provision is the *Integrated Services* architecture [2], which has been criticized in two ways. First, it must be widely deployed over the Internet to be useful, since its guarantees rely on actions at every router on a path. Second, it mandates resource reservations on a per flow basis, where a flow is a data stream between two user processes. Since a huge number of flows exists, the scalability of this scheme is questionable.

An alternative scheme that attempts to avoid these limitations is the *Differentiated Services* architecture [1], where flows are aggregated into a few classes, either when entering the network or when crossing network domains. At these points flows may be rate limited, shaped or marked to conform to specific traffic profiles. For neighboring domains, traffic profiles represent large traffic aggregates, while at network entry points they represent user requirements. Within a domain, routers only need to select a *Per-Hop Behavior* (PHB) for each packet, based on its class, denoted by the 8-bit Differentiated Services (DS) field of the IP header, which subsumes the IPv4 TOS and the IPv6 Traffic Class fields.

The services provided by this architecture are meant to provide generic QoS levels, not application specific guarantees. For example, the expedited forwarding PHB provides guaranteed bandwidth at each router, for traffic that was rate limited when entering the network or the domain so as not to exceed this bandwidth. This PHB provides low delay and loss by eliminating congestion for this restricted traffic class. Only network entry points are aware of both application requirements and PHB semantics so as to perform flow aggregation. Similarly, only domain entry points are aware of the semantics of PHBs available in their neighboring domains so as to perform appropriate translations.

Differentiated Services and multi-service link layers solve orthogonal but complementary problems. Differentiated Services are concerned with congestion and its impact on throughput, delay and loss. The services offered are link independent and are supported by IP level mechanisms. Multi-service link layers are concerned with recovery from link errors, customized to each type of application. The services offered are link dependent and local, as the frame scheduler only protects services from each other, it does not offer end-to-end guarantees. By only providing Differentiated Services over wireless links we can offer a nominal IP level QoS, but the actual performance will be limited by link losses. Multi-service link layers provide adequate recovery to fully utilize wireless links, but they need higher layer guidance to allocate link bandwidth.

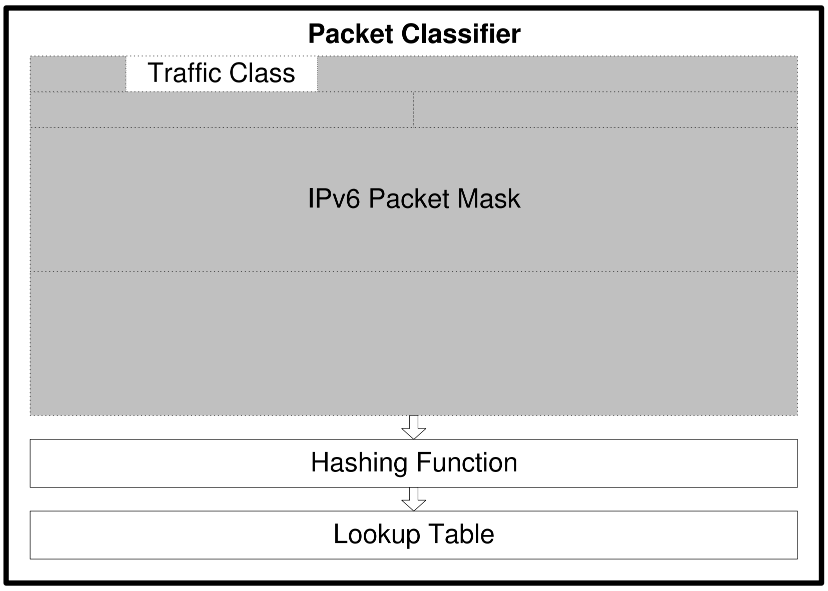

Fig. 12. Differentiated Services packet classifier.

These architectures can be implemented so as to complement each other. Differentiated Services provide congestion control, while multi-service link layers add application dependent error control. They both offer a few services (PHBs or link mechanisms) for traffic classes with common requirements. They can be combined by extending the DS field to also specify the error requirements of each traffic class. For example, a DS traffic class could be subdivided into two subclasses with different error recovery requirements by using one more bit of the DS field. Applications could indicate their requirements when injecting their traffic into the network, with boundary routers translating them to local equivalents. The result is a simplified classifier, shown in Fig. 12 for IPv6 packets. The DS field is isolated via a header mask, and then a hashing function plus a lookup table map it to an appropriate service.

This classifier does not rely on multiple header fields and complex rules to determine application requirements. More importantly, the DS field is not obscured by IP security mechanisms. Since the Differentiated Services module performs scheduling, traffic entering the multi-service link layer already obeys the required bandwidth allocations. Therefore, the subclasses of different DS classes can share the same link service without introducing congestion. The service rates can be set in two ways. If the subclasses of each DS traffic class have separate bandwidth allocations at the IP level, then the bandwidths for all subclasses mapped to the same link service are added to set its service rate. Otherwise, a measurement module can be inserted after the lookup table, as in Fig. 11, to determine service rates as explained in Sec. VII.

IX. Advanced Quality of Service Interface

The performance of the services available at each wireless link can greatly vary, depending both on the underlying link and on transient conditions. If we could provide a characterization of the end-to-end performance of a network path, applications could verify if a given service was suitable for their needs [31]. If these characterizations were updated whenever path characteristics changed significantly, they would also enable adaptive protocols and applications to modify their policies accordingly. To describe the services offered over diverse network paths however, we must dynamically discover what is provided at each multi-service wireless link on the path. This can be achieved if each service dynamically characterizes its performance with a set of standardized metrics. Dynamic characterization means that performance may be evaluated as often as needed, while metric standardization means that higher layers will be able to assess the performance of arbitrary services without any knowledge of the mechanisms employed. This would enable end-to-end QoS modules to compose local link metrics into end-to-end path metrics.

To support such services, we have defined three link independent metrics, reflecting the possible trade-offs available to error recovery schemes:

- *Goodput* (g_i, for service i) is the ratio of higher layer data transmitted during the measurement interval to link layer data transmitted, including all overhead. Note that the amount of higher layer data transmitted may differ from the amount received due to residual losses. Goodput can be calculated at the sender without any receiver feedback.
- *Loss* (l_i) is the ratio of higher layer data lost to higher layer data transmitted (lost plus received). Loss is calculated by the receiver based on the sequence of data released to higher layers. Residual loss can be greater than zero for limited recovery schemes.

TABLE I
SERVICE CHARACTERIZATION METRICS.

Name	*Symbol*	*Definition*
Goodput	g_i	$\frac{\text{higher layer data transmitted}}{\text{link layer data transmitted}}$
Loss	l_i	$\frac{\text{higher layer data lost}}{\text{higher layer data transmitted}}$
Delay	d_i	Average packet delivery delay
Effective Goodput	$e_i = g_i * (1 - l_i)$	$\frac{\text{higher layer data received}}{\text{link layer data transmitted}}$

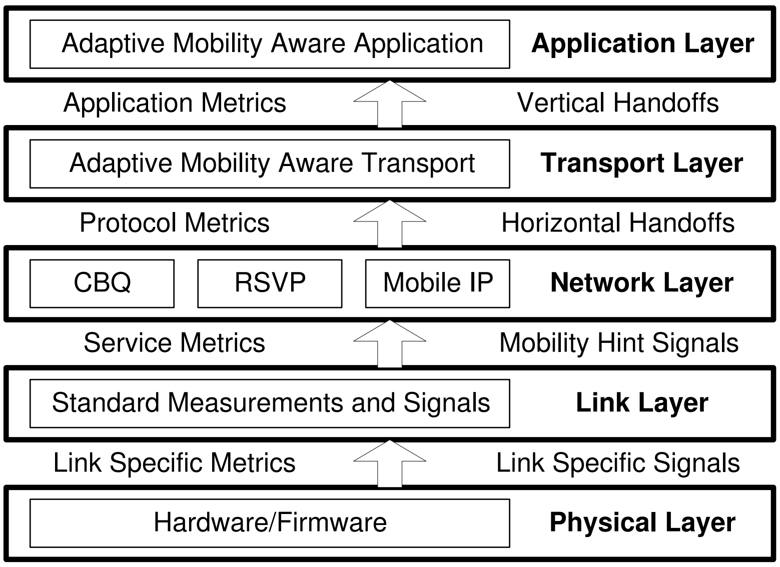

Fig. 13. Propagation of service measurement and mobility feedback.

- *Delay* (d_i) is the one way average delay for higher layer packets using this service. Delay can be estimated at the receiver based on knowledge of the implemented recovery scheme and wireless link characteristics. For example, retransmission schemes could add one round trip delay for each actual retransmission to their one way delay estimate.

The delay metric denotes error recovery delay only, so it should be added to IP level queueing delay to give the total delay for the node. Delay sensitive traffic subclasses should choose the lowest delay service whose residual loss falls within their tolerance limits. Goodput can be combined with loss to get *Effective Goodput* (e_i), defined as $e_i = g_i * (1 - l_i)$, or, the ratio of higher layer data received to link layer data transmitted. Essentially, e_i shows how much of the bandwidth allocated to a service is used by the data actually received, after subtracting error recovery overhead and residual losses. If the link bandwidth is B and service i is allocated a normalized service rate r_i in the scheduler, its throughput is $B * r_i * e_i$. This expression may be used to estimate the throughput for each service given a set of service rates, or to calculate the service rate needed to achieve a target service throughput. All metrics are summarized in Table I.

These metrics may be used at multiple layers to serve different needs, as shown in Fig. 13. The physical layer provides hardware information, such as the fixed one-way delay, that may be used by link layer services to provide their link independent metrics. At the network layer, scheduling mechanisms such as *Class Based Queueing* (CBQ) may use those metrics to set bandwidth allocations for each service. End-to-end QoS schemes may use the *Resource ReSerVation Protocol* (RSVP) to gather information about node services so as to estimate end-to-end path characteristics. These can in turn be used by transport protocols and applications to adapt their operation to prevailing conditions. For example, video conferencing applications may select encoding schemes appropriate to the residual loss characteristics of the path.

To assist higher layers in dealing with mobility, we can extend this interface to provide *mobility hints* to interested parties via upcalls, as shown in Fig. 13. The link layer can combine hardware signals with its own state to detect events like connections and disconnections. These link independent upcalls can be used by IP mobility extensions to allow fast detection of handoffs, instead of relying on periodic network layer probes [32]. Higher layers may be notified by the network layer via further upcalls of horizontal and vertical handoffs, i.e. handoffs between radio cells employing the same technology and handoffs between radio cells employing different technologies, respectively. TCP may be notified of pending horizontal handoffs to temporarily freeze its timers and avoid timeouts during disconnection. A video conferencing application may be notified of vertical handoffs so as to change the encoding scheme used to a higher or lower resolution one, depending on available bandwidth.

This interface fits the needs of *One-Pass With Advertising* (OPWA) [31] resource reservation mechanisms. One-pass schemes cannot specify a desired service in advance as they do not know what is available on a path [30], thus the resources reserved may prove to be inadequate. Two-pass schemes specify a service in advance but make very restrictive reservations on the first pass, relaxing them on a second pass. Such reservations may fail due to tight restrictions on the first pass. In an OPWA scheme, an advertising pass is first made to discover the services available on the path, and then a reservation pass reserves the resources needed. Our interface allows OPWA schemes to discover the restrictions imposed by wireless links. After that information is gathered, applications choose a service and the reservation pass sets up appropriate state to provide it. Mobility hints notify higher layers that they should revise path characterizations after handoffs, so as to support mobility aware applications.

X. SUMMARY

While extending the Internet over wireless links is straightforward, application performance over wireless links using the Internet protocols is often disappointing due to channel impairments. These problems considerably complicate the provision of Quality of Service guarantees over wireless Internet links. In order to investigate application performance over wireless links, we have simulated a number of link layer error control schemes. Our results show that different applications favor different approaches, so we developed a link layer architecture that supports multiple services simultaneously over a single link, allowing the most appropriate link service to be used by each traffic class.

Our approach is easy to optimize for each underlying wireless link and efficient to operate. It can be locally deployed, transparently to the rest of the Internet, and it is easy to extend to address future requirements. It can be incorporated into the Internet QoS architecture in three stages. First, in the current best-effort only Internet, heuristic classifiers can be employed to select the services provided over individual multi-service links, so as to enhance the performance of recognized applications. Second, in the context of the Differentiated Services architecture, which focuses on end-to-end congestion control, multi-service link layers can provide application dependent error control, respecting higher layer scheduling decisions despite the introduction of error recovery overhead. Finally, multi-service link layers can support an advanced QoS application interface, offering dynamic end-to-end service discovery.

REFERENCES

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

[2] D. D. Clark, S. Shenker, and L. Zhang, “Supporting real-time applications in an integrated services packet network: architecture and mechanism,” in *Proc. of the ACM SIGCOMM ’96*, August 1996, pp. 243–254.

[3] W. Stevens, “TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms,” RFC 2001, January 1997.

[4] 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.

[5] G. T. Nguyen, R. H. Katz, B. Noble, and M. Satyanarayanan, “A trace-based approach for modeling wireless channel behavior,” in *Proc. of the Winter Simulation Conference*, 1996.

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

[7] 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.

[8] R. Ludwig and R. H. Katz, “The Eifel algorithm: making TCP robust against spurious retransmissions,” *Computer Communications Review*, vol. 30, no. 1, pp. 30–36, January 2000.

[9] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, “TCP selective acknowledgment options,” RFC 2018, October 1996.

[10] 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.

[11] S. Kent and R. Atkinson, “IP encapsulating security payload (ESP),” RFC 2406, November 1998.

[12] 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.

[13] J. Liu and S. Singh, “ATCP: TCP for mobile ad hoc networks,” *IEEE Journal on Selected Areas in Communications*, vol. 19, no. 7, pp. 1300–1315, July 2001.

[14] S. Nanda, R. Eljak, and B. T. Doshi, “A retransmission scheme for circuit-mode data on wireless links,” *IEEE Journal on Selected Areas in Communications*, vol. 12, no. 8, pp. 1338–1352, October 1994.

[15] 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.

[16] Q. Pang, A. Bigloo, V.C.M. Leung, and C. Scholefield, “Performance evaluation of retrnasmission mechanisms in GPRS networks,” in *Proc. of the IEEE WCNC ’00*, 2000, pp. 1182–1186.

[17] R. Ludwig and B. Rathonyi, “Link layer enhancements for TCP/IP over GSM,” in *Proc. of the IEEE INFOCOM ’99*, 1999, pp. 415–422.

[18] C. Parsa and J.J. Garcia-Luna-Aceves, “Improving TCP performance over wireless networks at the link layer,” *Mobile Networks and Applications*, vol. 5, no. 1, pp. 57–71, 2000.

[19] I. Aad and C. Castelluccia, “Differentiation mechanisms for IEEE 802.11,” in *Proc. of the IEEE INFOCOM ’01*, 2001, pp. 209–218.

[20] M. Barry, A. T. Campbell, and A. Veres, “Distributed control algorithms for service differentiation in wireless packet networks,” in *Proc. of the IEEE INFOCOM ’01*, 2001, pp. 582–590.

[21] The VINT Project, “UCB/LBNL/VINT Network Simulator - ns (version 2),” Available at http://www.isi.edu/nsnam.
[22] G. Xylomenos, “Multiservice link layer extensions for ns-2 (version 1),” Available at http://www.mm.aueb.gr/˜xgeorge/codes/codephen.htm.
[23] G. Xylomenos and G. C. Polyzos, “Multi-service link layer enhancements for the wireless Internet,” in *Proc. of the 8th IEEE Symposium on Computers and Communications*, 2003.
[24] G. Xylomenos and G. C. Polyzos, “Wireless link layer enhancements for TCP and UDP applications,” in *Proc. of the 3rd International Workshop on Wireless, Mobile and Ad Hoc Networks*, 2003.
[25] 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.
[26] B. A. Mah, “An empirical model of HTTP network traffic,” in *Proc. of the IEEE INFOCOM ’97*, 1997, pp. 592–600.
[27] S. Nanda, D. J. Goodman, and U. Timor, “Performance of PRMA: a packet voice protocol for cellular systems,” *IEEE Transactions on Vehicular Technology*, vol. 40, no. 3, pp. 584–598, August 1991.
[28] G. Xylomenos and G. C. Polyzos, “Quality of service support over multi-service wireless Internet links,” *Computer Networks*, vol. 37, no. 5, pp. 601–615, 2001.
[29] S. Golestani, “A self-clocked fair queueing scheme for broadband applications,” in *Proc. of the IEEE INFOCOM ’94*, 1994, pp. 636–646.
[30] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, “RSVP: A new resource reservation protocol,” *IEEE Network*, vol. 7, no. 5, pp. 8–18, September 1993.
[31] S. Shenker and L. Breslau, “Two issues in reservation establishment,” in *Proc. of the ACM SIGCOMM ’95*, October 1995, pp. 14–26.
[32] C. Perkins, “IP mobility support,” Internet Request For Comments, October 1996, RFC 2002.