DIRAC: A Software-based Wireless Router System

Petros Zerfos, Gary Zhong, Jerry Cheng, Haiyun Luo, Songwu Lu, Jefferey Jia-Ru Li
UCLA Computer Science, Los Angeles, CA 90095.
Email: {pzerfos, gzhong, chengje, hluo, slu}@cs.ucla.edu, juru@hiq.com

ABSTRACT

Routers are expected to play an important role in the IP-based wireless data network. Although a substantial number of techniques have been proposed to improve wireless network performance under dynamic wireless channel conditions and host mobility, a system support framework is still missing. In this paper, we describe DIRAC, a software-based router system that is designed for wireless networks to facilitate the implementation and evaluation of various channel-adaptive and mobility-aware protocols. DIRAC adopts a distributed architecture that is composed of two parts: a Router Core (RC) shared by the wireless subnets, and a Router Agent (RA) at each access point/base station. RAs expose wireless link-layer information to the RC and enforce the control commands issued by the RC. This approach allows the router to make adaptive decisions based on link-layer information feedback. It also permits the router to enforce its policies (e.g., policing) more effectively through underlying link-layer mechanisms. As showcases, we implement under DIRAC the prototypes of three wireless network services: link-layer assisted fast handover, channel-adaptive scheduling, and link-layer enforced policing. Our implementation and experiments show that our distributed wireless router provides a flexible framework, which enables advanced network-layer wireless services that are adaptive to channel conditions and host mobility.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Design, Performance, Experimentation

Keywords

Distributed Router Architecture, Wireless Network

MobiCom'03, September 14–19, 2003, San Diego, California, USA.

1. INTRODUCTION

Routers have been a key architectural component in the IP-based wired Internet. As wireless data services become increasingly popular and wireless networks move toward an IP-based paradigm, routers are expected to play an equally important role in the emerging wireless Internet era.

The main functionality of a router is data-plane packet forwarding and control-plane routing and management [1, 2]. The evolution of the Internet has also made a case for new services such as packet filtering, intrusion detection, level-n switching, and packet tagging [2]. Despite all such advances, current router systems do not work well in wireless networks because they do not consider the issues of wireless link and host mobility.

In recent years, a large number of protocols [3, 4, 5, 6, 7] have been proposed to achieve the above goal. At the core of such protocols, they use link-layer information feedback, e.g., channel errors and link handoff events, to make channel-adaptive and mobility-aware decisions. These solutions have been carefully analyzed and evaluated through simulations. However, they have not been able to materialize in practice. The key missing piece is a system framework that facilitates the real implementation, experimental evaluation and field deployment of such new solutions. Developing a router system that provides such a system framework is the subject of this work.

This paper describes the design and implementation of DIRAC[1], a software-based wireless router system. DIRAC currently works with IEEE 802.11b, but its design principles apply to other wireless technologies such as 802.11a, 802.11g, and wide-area 3G and 4G. The key technical innovation of DIRAC is a distributed router architecture (shown in Figure 1), consisting of a *router core* (RC), and several *router agents* (RAs), each of which runs on an 802.11 access point (AP). The RC is shared by the wireless subnets and carries the main functionality of a router. It interacts with each RA through a lightweight communication protocol. Each RA collects and reports link-layer information back to the RC, and accepts and enforces router policies such as policing via link-layer mechanisms (e.g., Authentication/Reauthentication of 802.11 standard). The resulting benefits of such a design are twofold: (1) It enables the router to make *informed, adaptive* packet forwarding decisions on the data plane and improved mobility management on the control plane, based on link-layer information feedback. (2) It allows the router to directly execute its network-

[1]DIRAC denotes DIstributed Router ArChitecture for wireless networks.

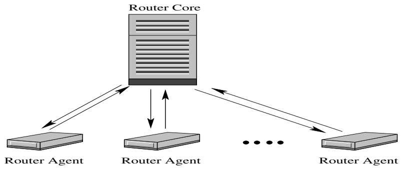

Figure 1: Distributed wireless router architecture

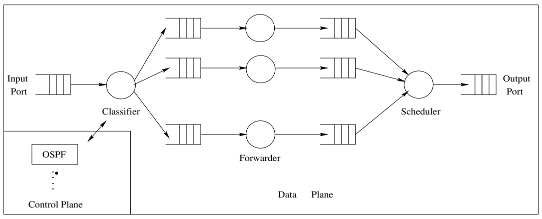

Figure 2: A Wired Router

level policies via the exposed link-layer mechanisms. As a result, DIRAC gains from the enhanced interactions with the link layer and facilitates the implementation, evaluation and deployment of wireless-specific protocols and services.

The distributed architecture of DIRAC is motivated by the fact that link-layer information, on per host basis, is location dependent and only accessible at each AP. Moreover, DIRAC stays in the middle ground of two extreme architectural alternatives: the intelligent AP approach that provides adaptive link-layer operations at each AP but remains network-layer oblivious, and the ubiquitous router approach, which converts every AP into a full-blown router, and enables all network-layer functions at the cost of increased complexity in management and administration. The DIRAC design allows for cost-effective and rapid development of wireless services by upgrading software only at the RC, without modifying the hardware and/or software of each AP. It facilitates monitoring and configuration of the network, since most complexity is placed on the RC, and enhances robustness by keeping each access point simple. Last, DIRAC not only simplifies the implementation and evaluation of adaptive, wireless protocols designed for a single cell, but also permits the deployment of inter-cell coordinated resource management protocols.

With the architecture in place, we have implemented several prototype services to demonstrate its viability and practicability. The three services implemented are wireless and mobility specific, and include a link-layer informed, fast handover protocol on the control plane that minimizes transient packet losses during handoffs, a channel-adaptive, FEC-based, packet forwarding solution that solves the head-of-line blocking issue over wireless links, and a link-layer assisted policing that penalizes overly aggressive, wireless clients. The list of these three example protocols demonstrates the wide spectrum of wireless router services enabled by DIRAC. Other services, such as inter-AP coordinated resource management, mobility-aware firewall, adaptive wireless scheduling, and cross-cell QoS support for VoIP, can also be implemented.

The implementation of DIRAC architecture and services is developed using off-the-shelf hardware and open-source software. The router core is written in Click [8], an extensible and modular software router framework, and runs on a commodity PC. The router agents are implemented on the Instant AP platform [9]. Our experiments show that the system is able to support a large number of access points as well as clients per AP using commodity hardware: processing statistics reports from 50 APs requires only $140\mu secs$, and it takes less than $1\mu sec$ to process channel states for 50 clients per AP.

The rest of this paper is organized as follows. Background introduction is provided in Section 2. Motivation for DIRAC is given in Section 3. The design and implementation of the basic DIRAC framework are given in Sections 4 and 5, respectively. Section 6 describes three prototype services for wireless routers. Section 7 evaluates the system performance of DIRAC. Discussions and lessons learned are provided in Section 8. Section 9 compares with related work and the paper is concluded in Section 10.

2. BACKGROUND

The target scenario is a packet-switched wireless data network, based on the 802.11b wireless access technology. In a typical enterprise/campus environment, mobile hosts access the Internet via the access point (AP) in each cell over the wireless channel, and APs are interconnected via the wired backbone. Multiple cells form a subnet, several subnets together cover the entire enterprise/campus and are interconnected via an access router. Mobile hosts may roam between adjacent cells, possibly across different subnets. Packet transmissions can be downlink (from the network to a mobile host), or uplink (from a mobile host to the network). The scenario we focus on –a typical one in practice– assumes that the wired network has sufficient bandwidth, and the wireless link poses as the bottleneck.

A typical wired router (shown in Figure 2) consists of a data-plane forwarding engine that forwards packets, and a control plane where signaling protocols run. In the forwarding engine [1], a classifier first reads packets from an input port, and based on certain fields in the packet header, selects a forwarder to process the packet and send it to the output queue. Finally, an output scheduler selects one of its output queues and transmits the packet to the output port. The control plane of a wired router executes signaling protocols like OSPF and LDP and assists the packet forwarding engine in the process of packet delivery.

The above wired router system does not work well in a wireless network. The fundamental problem is that it does not consider the issues of dynamic wireless link quality and host mobility. The resulting performance degradation is well documented in the literature, in the context of wireless scheduling [4], adaptive error control [7], micro-mobility management [10], and admission control, to name a few.

3. DESIGN RATIONALE

3.1 Motivation

In recent years, the demand for wireless services, such as carrier-grade data delivery, security protection, and QoS support for delay-sensitive or communication-intensive applications including VoIP, interactive multiplayer gaming, and multimedia instant messaging, has significantly increased. In order to offer such wireless-specific services, numerous protocol solutions have been devised. Examples include adaptive error control mechanisms, wireless packet scheduling, mobility-aware admission control, firewall for roaming users, micro-mobility management protocols, adaptive bandwidth reservation for VoIP, etc. The fundamental principle behind such wireless protocol proposals is to adapt to wireless channel dynamics and also be mobility aware. The design tenet is to allow for closer but constrained interactions with the underlying link-layer to improve protocol performance and adaptability. In order to enable the implementation and deployment of such wireless networking protocols, a system support framework that enhances this interaction between the edge router and the network access technology is needed.

Some of the aforementioned wireless services, such as fast handoffs for micro-mobility management, admission control for roaming users, and adaptive reservation for mobile VoIP calls, require not only closer interaction with the link layer in a single cell, but also inter-cell coordination across multiple APs. The fundamental problem is that roaming users incur resource dynamics in both the time and the spatial domain. Thus, wireless resource management schemes have to coordinate decisions among neighboring cells. The resulting benefits not only enable seamless services for mobile users, but also minimize the inter-cell channel interferences. DIRAC seeks to enable such inter-cell coordinations.

Furthermore, a typical enterprise or campus wireless network needs a large number of APs (up to several hundreds) to cover the entire service area. This fact poses the challenge of minimizing the software and hardware upgrade cost, and the associated management overhead, when deploying new services. The solution lies on intelligently partitioning the software between the access points and the access router, so as to place most of the complexity at the centralized router, while keeping the distributed software module at each AP simple and generic. DIRAC fulfills this requirement.

Finally, recent trend in Internet edge router design advocates the use of software-based frameworks, which provide extensibility and flexibility. The software-based router accelerates the implementation, experimentation and deployment of new network protocols and algorithms. The potential performance penalty can be offset by combining this approach with network processor cards. DIRAC also follows this trend and takes the software-based router design approach. In some sense, flexibility of the access router system is more important than mere data-forwarding speed in the wireless domain because of the fundamental limit of wireless channel capacity.

3.2 Architecture Alternatives

The design of DIRAC stays in the middle ground of several extreme architectural options.

The first option is the oblivious approach followed by current practice, in which the edge router remains oblivious of the wireless characteristics, and the AP delivers 802.11 MAC functionality only. However, this suffers from severe performance degradation as exemplified by the head-of-line blocking problem over the wireless channel in Section 6.2.

The next alternative is the intelligent AP approach, in which adaptive link-layer services are implemented at each network-layer oblivious access point. The shortcomings of such an approach are also well understood: network-layer functions that require knowledge and handling of the IP characteristics of the packets, in order to distinguish flows for admission control, identify subnets for fast handoffs between cells, and measure bandwidth for reservation purposes, cannot be enabled with link-layer mechanisms only. Moreover, this approach may also need high-end hardware on each AP to run processor-intensive tasks. For example, the achievable throughput for the proposed FEC service (see Section 6) on an i486 processor at 66MHz is around 4.05Mb/sec, while the AP processor is even less powerful (it is an AMD SC400 at 33MHz in our testbed). The throughput offered by the current AP hardware is not sufficient for an 11Mbps 802.11b network, and certainly inadequate for 802.11a/g. Furthermore, interoperability issues among different network access technologies such as 802.11 and 3G, 802.11b and 802.11a, also arise.

Another extreme alternative is the ubiquitous router approach, which converts each AP into a router. It also exhibits several downsides. Since network routers are far more complex devices, the management and monitoring of such a large number of routers (e.g. 50 to 100 in a typical enterprise subnet) become problematic. It incurs considerable workload for the administrator, and the deployment of new services is nontrivial. A large number of nodes have to be upgraded, both in hardware and software, each time a new service is deployed. Moreover, it still does not enable inter-cell coordination needed for some wireless services; inter-cell information is still not available at a centralized point. Furthermore, since there is a large number of devices that need to be replicated, and this task incurs considerable administrative and deployment cost.

From the above, it is clear that DIRAC arbitrates the extreme architectural options previously discussed. It seeks to preserve the best features and avoid the drawbacks of each approach, and balance between performance, complexity, cost and manageability.

4. DESIGN

4.1 Overview

DIRAC adopts a distributed router architecture that consists of two main software components: a generic Router Core (RC), and multiple lightweight, network-specific Router Agents (RAs). The architecture is distributed in the sense that each RA runs at an AP, while the RC runs on a commodity PC. They collaborate to provide router functions and services – data-plane packet forwarding and control-plane management – to each mobile host. The RC interacts with the wireless link layer through each RA, without changing the functionality of the IP layer. The interaction between RC and RA is constrained; it takes three forms: *events*, *statistics* and *actions*, to be elaborated in Section 4.2.

The RC carries out the regular operations of a router for each wireless subnet it connects. It has a packet forwarding engine and a control plane. The forwarding engine is renovated to address wireless link issues. It accepts link-layer information feedback, in the form of channel *statistics* of each mobile host from the RA, and, based on such information, it performs adaptive forwarding operations. The data forwarding plane also requests the RAs for *actions.* Such actions will be executed using link-layer mechanisms. The control plane of the RC accepts *events* from the RA. These may in turn trigger management operations on the RC. A DIRAC control engine has been added to the RC, which arbitrates the flow and dissemination of *events*, *statistics*, and *actions*.

The Router Agent is link-layer specific and light-weight. It does not perform conventional router forwarding and management functionality. Instead, it serves as a messenger between the RC and the link-layer device driver. Communication between the RC and the RA is carried out via standard UDP sockets over the wired backbone.

4.2 Forms of Interaction

The interaction of DIRAC and the underlying 802.11 link layer takes three forms: *events*, *actions*, and *statistics*. Table 1 illustrates the typical events, statistics and actions implemented by the DIRAC router for the 802.11b technology. It shows a representative set of primitives supported by most wireless access technologies.

Events denote occurrences of asynchronous link-layer activity in the cell, detected by the access point and reported back to the RC control engine. For example, a Reassociation event informs on a layer-2 handoff by a roaming host. DIRAC acts upon events to make mobility-aware decisions.

Statistics report the latest information on channel quality that each host perceives, which is location-dependent. Such information can be expressed in link-layer frame loss percentage, or even SNR (Signal-to-Noise Ratio) in dB. DIRAC uses such statistics to make channel-adaptive packet delivery via its renovated scheduler or forwarder.

Actions are requested by RC in order to enforce its policies. Router Agents, which receive these requests, execute them at each AP's link-layer, either by tuning the parameters of the 802.11b MAC protocol, or by adjusting the configuration settings of the driver. For example, *Deauthentication* action will trigger the AP to deny the channel access by the target host; *Set_Retransmissions* action will ask the AP to reset the maximum retransmission count.

Events	*Actions*	*Statistics*
New host (Association)	Accept/Reject host Association	Signal-Noise Ratio (dB)
Host Leaves (Disassociation)		Channel Quality Variation
Security Binding (Authentication)	Authentication Deauthentication	Frame Retxmits (frame loss %)
Roaming host (Reassociation)	Set maximum retransmit count	CommTallies
Power Saving mode (PS)		Latency

Table 1: Events, Actions, and Statistics

4.3 Router Core

The Router Core renovates its data-plane forwarding engine and control-plane management protocols to make them wireless adaptive and mobility aware.

4.3.1 Control plane

The control plane of DIRAC consists of two types of components: routing and management protocols, and a *control engine*. DIRAC does not mandate a specific choice of the former and leaves it to the users. The control engine in DIRAC is the OS support to enable the cross-layer interactions via *events*, *statistics* and *actions*. It consists of four components: *EventProcessor*, *StatisticsMonitor*, *ActionProcessor*, and *RegistrationDB*.

EventProcessor accepts messages carrying events reported by each RA and notifies each interested party of such events. Any component on the data plane (e.g., the forwarder) or on the control plane (e.g., the fast handover module) can request and will be informed of a specific event upon its occurrence. This functionality is provided by a *callback* mechanism. Each module interested in an event provides a callback function handler to the EventProcessor, one for each event type. Upon arrival of an event notification, its registered callbacks are executed.

StatisticsMonitor provides a centralized repository to maintain the latest channel quality information for each host. Other elements of the router can query it, retrieve the updated link-layer statistics, and adapt their operations accordingly. The statistics information is updated periodically via messages reported by RAs to the RC over the wired connection. To minimize the memory overhead, we store only the latest received report, along with a timestamp.

ActionProcessor sends an *action_request* message to the appropriate RA, upon request from any module within the RC. It serves as an interface between the RC and the link-layer AP. It encapsulates each action request into a UDP message, and sends it to the corresponding RA.

Finally, *RegistrationDB* offers a registry to store information regarding each mobile host and its associated AP. Each host is assigned a unique ID throughout the router system. This way, even though different underlying network technologies may use different link-layer IDs, the router still treats them in a coherent way.

4.3.2 Data plane

The data-plane forwarding engine allows components such as schedulers and forwarders to implement channel-adaptive protocols, based on link-layer feedback. DIRAC does not stipulate the specific choice of such protocols; many solutions in the literature may serve this purpose.

A distinctive feature of the forwarding engine is that it provides *asymmetric* operations for *uplink* and *downlink* flows (shown in Figure 3). The downlink service is proactive and the service provided via wireless scheduler or forwarder is fine grained with a time granularity of tens to hundreds of milliseconds. The uplink service is reactive and the service provided via policing is coarse grained with a time granularity of seconds.

The above design choice is motivated by the inherent wireless network constraint. The router core has complete control and accurate flow information (e.g., the arrival time and length of each packet) for downlink flows. However, flow information for uplink flows is spread at each mobile host. The router core does not have full control and accurate information for uplink flows. Early research [3] takes the alternative approach of symmetric operations for both uplink and downlink flows. Its downside is that each host has to propagate its flow information back to the router core on a per-packet basis. This incurs heavy communication and processing overhead, which we seek to avoid.

The resultant asymmetric services for uplink and downlink also fit well with the wireless traffic characteristics. Current wireless traffic measurements [11] show that 66% of the traffic is downlink, largely due to applications that download data from Internet servers.

4.4 Router Agent

The RA at each AP bridges the interaction between the RC and the wireless link layer. The goal is not to facilitate generic programmability of the access point, but to

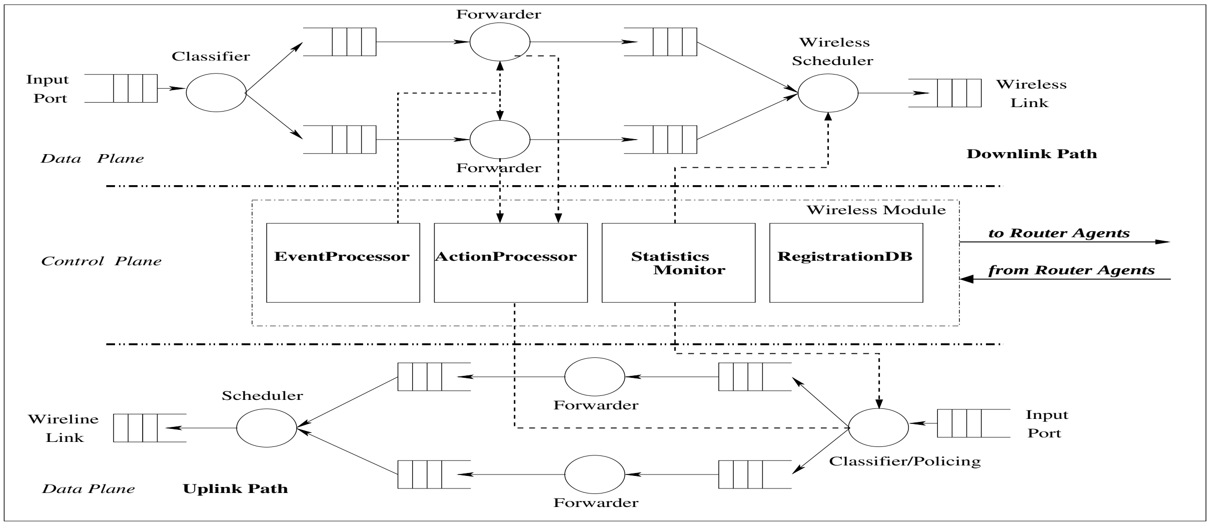

Figure 3: Conceptual model of the Router Core

run a light-weight messenger between layer-2 of the AP and layer-3 of the access router. Thus, its tasks include: (1) Monitoring the state and channel quality for each of its associated hosts, as reported by the device driver. (2) Sending appropriate messages to the RC, when events occur in the cell. (3) Intercepting messages sent by the RC, requesting *action* messages from the AP device driver, and delivering other messages to the mobile host in the form of link-layer frames.

Link-layer statistics are collected for each mobile host by the RA periodically. The collection frequency reflects a tradeoff between state accuracy and communication overhead. A rough calculation shows that the frequency need not be high, since the channel coherence time (i.e., the period during which channel conditions does not vary much) is about 50-100ms for walking-speed hosts.

4.5 Architectural Features

The cost saving provided by DIRAC comes from "economies of scale" in two aspects: hardware investment, and "soft-cost" associated with management and administration. The former is relatively easy to assess, while the latter is harder to gauge quantitatively. For the hardware, only upgrading the access router is needed in DIRAC each time a new service requires more computational power. This can be done by using network processor cards. For example, using the Intel IXP 1200 network processor cards for a moderate US $700 per piece, one can parallelize operations and aggregate services while providing high performance (up to 5Gbps). On the other hand, upgrading each AP can be much more expensive if one does not use DIRAC. Though each AP typically costs about US $150, one has to take into account that a large number of them (in the order of 50 to 100 per access router, according to a study by Juniper Research [12] for campus-sized environments) will have to be upgraded. Moreover, since the centerpiece is placed at the RC in DIRAC, centralized management for subnets is made possible, which in turn reduces the administration overhead.

DIRAC also takes an asymmetric approach to functional partition between the RC and each RA. RAs are controlled by the RC to certain extent, while wireless network services are centrally implemented at the RC. Maximum programmability is allowed at the RC, but not at RAs. In addition, the Router Core is rather link-layer technology independent. When applied to a new link-layer technology, it can be kept intact and only updating the lightweight RAs is needed. This improves portability of the system to other wireless technologies.

5. IMPLEMENTATION

This section describes DIRAC implementation, including how the RC and each RA work together in the specific hardware and software environment. The RC is about 5900 lines of code, and the RA is about 1000 lines.

5.1 Router Core

The RC works with the IPv6 protocol on a commodity PC. We choose IPv6 as our working protocol environment because its extensibility and inherent mobility support have been well articulated in the literature [13]. In fact, the cellular network industry has been favoring Mobile IPv6 over Mobile IPv4. Our implementation is therefore based on IPv6, but it can be easily ported to IPv4.

RC is implemented as a set of *Click elements* within the Click router framework [8], under Linux. We chose Click as the implementation platform for the RC, since its extensible and modular architecture allows for rapid prototyping. Click also provides standard functionality of the forwarding path, such as routing table and address resolution. For DIRAC, we developed 13 new Click elements (Table 2 provides a glossary, along with a short description) that implement the desired functionality. Unlike most existing Click elements, they are shared and comparatively larger, due to the complicated operations that they carry out.

Figure 4 shows a two-interface (one wireless, and one wired) router core configuration with the Click elements. *IP6FastHandover* and *DownlinkScheduling* are two compound Click elements to be described in Section 6. The downlink and uplink packet forwarding engines are shown in dotted lines in the figure.

Class	*Element*	*Description*
Control Engine	EventProcessor	Processes event-messages from RAs; invokes callback functions of other elements
	ActionProcessor	Encapsulates actions requested by other elements into UDP messages sent to RA
	IP6L2Statistics	Stores statistics about each mobile host. Statistics are sent as UDP messages by RA
	RegistrationDB	Stores matchings between hosts and RAs; stores addresses, and assigns IDs
Mobility Management	IP6MobilityNewAR	Implements the micro-mobility management protocol –new AR (Section 5.1)
	IP6MobilityOldAR	Implements the micro-mobility management protocol –old AR (Section 5.1)
	IP6Tunnel	Implements IPv6-IPv6 tunneling as virtual interfaces
	IP6Buffer	Buffers packets, as it waits for a Fast Neighbor Advertisement message to arrive
	IP6FNDSolicitor	A neighbor solicitor for IPv6 that also handles Fast Neighbor Advertisements
	IP6RAdvertiser	Sends Router Advertisements, both solicited and unsolicited (also see [14])
Downlink path	FECEncoder	Performs packet FEC enconding on k packets, producing $n-k$ redundant ones
	FECController	Controls the *FECEncoder* by setting k and n according to the channel quality
Uplink path	PolicingMeter	Police aggressive uplink flows

Table 2: Element Glossary

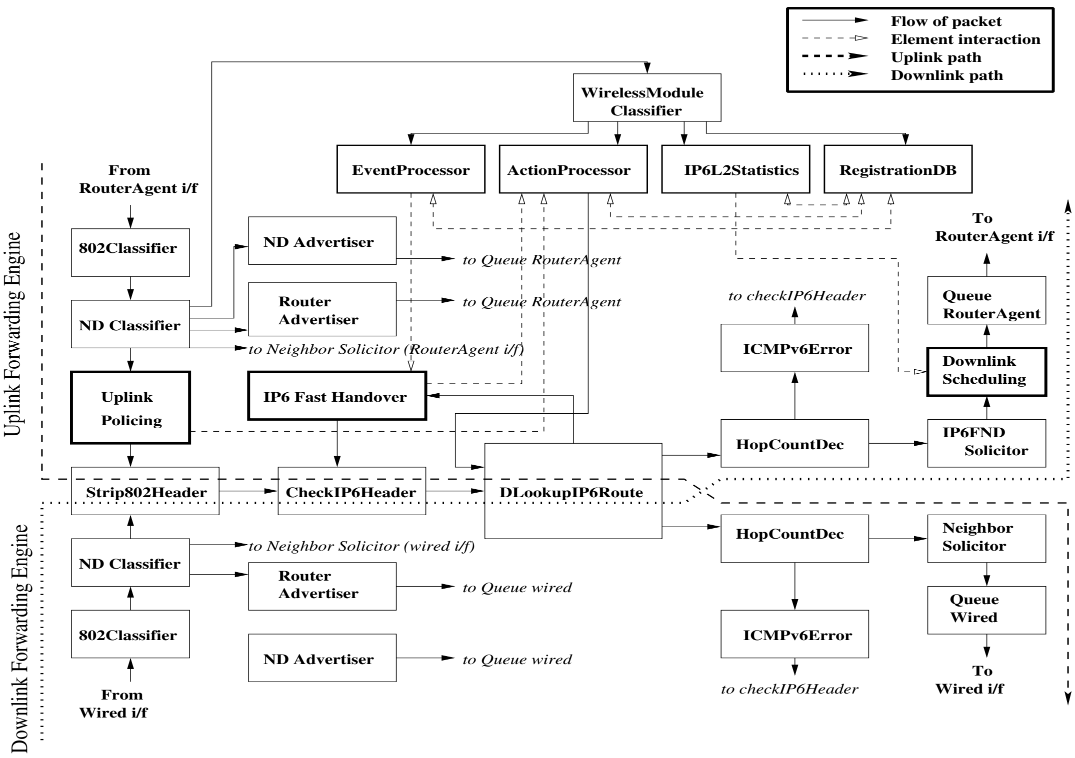

Figure 4: *Click* **element configuration diagram for the Router Core.**

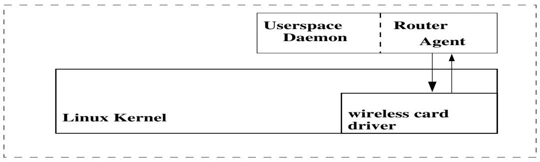

Figure 6: Implementation of the Router Agent.

`execute`(`out` status-code, `in` node-id, `in` action-type, `in` params)
(a) Request for Action (*ActionProcessor*)

`cancel`(`in` request-id)
`register`(`in` event-type, `in` element-name, `in` callback-function, `out` request-id)
(b) Registration for Events (*EventProcessor*)

`element.callback`(`in` request-id, `in` event-type, `in` params)
(c) Callback Function (*EventProcessor*)

`get_stat`(`in` node-id, `in` stat-type, `out` value)
(d) Retrieval of Statistics for link-quality (*IP6L2Statistics*)

`get_assoc`(`inout` node-id, `inout` router-agent-id)
`get_node`(`in` node-id, `out` ip6-addr, `out` ll-addr-type, `out` ll-addr)
(e) Queries to the *RegistrationDB*

Figure 5: API provided by DIRAC control engine

The *downlink* forwarding engine operates as follows. Incoming packets from the device interface go through the 802.1d frame classifier *802Classifier* to check whether it is an IPv6 neighbor discovery message [14], followed by the neighbor discovery classifier *NDClassifier* to check whether it is a router advertisement message. The 802.1d header is stripped by the *Strip802Header*, and the remaining IPv6 header is examined by *CheckIP6Header*. *DLookupIP6Route* performs routing table lookup for the packet, *HopCountDec* decrements its hop count, and passes it on to the *FNDSolicitor* which handles the address resolution. It then goes through the *Downlink Scheduler* and finally is delivered to the interface where the Access Point/Router Agent is connected. The *uplink* forwarding engine works similarly except that it goes through an *Uplink Policing* element first.

5.1.1 API

The aforementioned wireless network services, as well as new ones to be deployed in the future, make use of an API provided by the control engine of the Router Core. Each service consists of its own elements (in the Click framework), but all services use the same API to interact with the RC. Four elements of DIRAC (*EventProcessor, Action Processor, IP6L2Statistics, and RegistrationDB*) implement and make this interface available to the other elements of the Click configuration.

Taking actions Elements take link-layer actions using the API provided by the `execute` call of Figure 5 (a). The *execute* call takes input on which host requests the call, the type of action requested, and the parameters for the action type, and returns a status code of failure/success. The *ActionProcessor* element, which actually implements the action, looks up the requesting host's link-layer address and the respective RA using the API provided by *RegistrationDB* of Figure 5 (e). If the action is not targeted to a mobile host (for example, it is to set retransmissions on/off on the particular AP), the *node-id* parameter is discarded.

Notifying elements In order for elements to be notified upon an event that happened in the cell, they must register first with the *EventProcessor* via the `register` API of Figure 5 (b). The *register* call lets the element specify the type of events it is interested in, its name, and a callback function. *EventProcessor* subsequently returns a unique ID for that request. The callback function is then stored in a linked list, one for each type of event.

Once *EventProcessor* receives an event notification message from an RA, it invokes the callback mechanism, which provides exactly-once, in-order semantics for each element to be notified of a concrete event. Specifically, it traverses the linked list and invokes the *callback* call of Figure 5 (c). The call specifies who requests callback, the type of events for callback, and event parameters.

Monitoring link quality An element learns of the channel statistics each host perceives by invoking the `get_stat` call of Figure 5 (d). The call asks for the host ID, the type of statistics it is interested in, and returns the current channel statistics. The current implementation returns only a single normalized value for channel quality computed by the RA. We plan to enrich the statistics with more elaborate information in the near future.

In the current implementation, elements learn the statistics via queries. A more proactive mechanism could follow the approach of [15]: Interested elements specify the lower and upper bounds for the value of a statistic; *IP6L2Statistics* element then triggers callback handlers when the statistic falls into these predefined bounds.

5.2 Router Agent

The RA is implemented on the OpenAP platform [9]. The hardware consists of a WL11000 SA-N board, which is equipped with a wired Ethernet controller (NE2000), an AMD ELAN SC400 embedded processor running at 33MHz, 1MB of Flash RAM, and a RS-232 serial interface for the output of the console. The wireless PCMCIA 802.11b network card is made by U.S. Robotics, using Intersil Prism2 chipset. The hardware platform is programmable through a relatively simple process; it runs an embedded version of Linux 2.4.17 in our implementation.

The Intersil chipset offers a "Host AP" operation mode. In this mode, the card and the host driver act as an AP for mobile clients and provide all management and control functionality of the 802.11 standard. The firmware handles time-critical tasks such as beaconing and frame acknowledgment, while the management functionality is provided by the host driver. Part of the driver, which provides the interface to the card, runs as a kernel module, and the remaining piece that handles the 802.11 management sublayer runs as a user-space daemon [16]. The user-space daemon opens a raw socket to the device that appears as a virtual interface to Linux. Through this socket, it is able to transmit and receive management frames.

The RA (see Fig. 6) is implemented as part of the host computer driver [16], and extends both of its kernel- and user-level components. To collect the link-layer statistics, we implemented an *ioctl()* call in the kernel driver that updates variables in the driver from the hardware configuration records of the card. A more efficient implementation that would also eliminate context switches, would update these values using task_queues [17]. However, for the purposes of our experiments, and since statistics are reported every 100ms –sufficient for capturing the behavior of the channel for a user moving at walking speed – the solution with the ioctl() was preferred. The user-level part of the RA periodically calls this ioctl(), and the results are sent to the RC via the wired network.

The user-space component of the RA also registers with the event-loop of the driver. It intercepts management frames, recognizes events, and sends event notifications as UDP messages to the core. Similarly, actions received in the form of messages from the RC are interpreted in a hardware-specific manner by the RA. Either customized 802.11 management frames are sent to the mobile host (e.g. Re-association reply frame), or configuration settings of the card are changed (e.g. MAC access lists).

5.3 RC↔RA Communication Protocol

RC and its associated RAs communicate through a simple protocol implemented in UDP, which runs over the wired backbone. This protocol is used for exchanging information that enables the two-way interaction. The packets exchanged between the RAs and the RC follow the generic TLV (Type-Length-Value) format, enhanced with a subtype field:

```
-----------------------------------------
|  Type   | Subtype| Length|  Value ...  |
-----------------------------------------
```

In the above-defined packet, The "Type" field specifies the type of interaction that the packet carries, and can be one of the four types: Statistics, Event, Action, or Registration of an RA to RC. The "Subtype" field specifies the specific subtype of Statistics, Action, Event, or Registration. For example, for Statistics on channel quality, the subtype can be SNR, Frame_Loss, Latency, or Variance. The "Length" field denotes the size (in bytes) of the "Value" field. It is used when multiple values of the subtype of the interaction are carried in the same packet, such as statistics reports for multiple clients. Finally, the "Value" field carries the actual information. It is subtype-specific and depends on the subtype of the interaction. For example, the "Value" field for statistics on each node can be:

```
-----------------------------------------------------
| Mobile Node LL Addr. |  Measurement | Interval |
-----------------------------------------------------
```

where the first field is the link-layer (Ethernet) address of the mobile node, the second field is the measured statistics value, and the third field is the interval during which this value was calculated.

The "Value" field for actions follows the format:

```
------------------------------------------
| Mobile Node Link Layer Addr. |  Code   |
------------------------------------------
```

where the first field is the link-layer address of the mobile node upon which the action is to be taken, and the second field stipulates the action to be taken.

The "Value" field for events also has a subtype-specific format, although in most cases it follows the format of the action, as shown above. However, in certain cases, it may take a richer form, such as in the event of a roaming host, where the old AP is also provided:

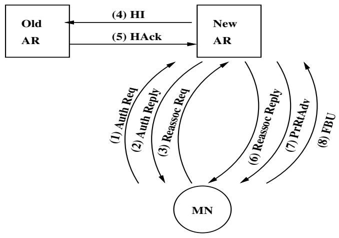

Figure 7: Link-layer informed fast handover

```
----------------------------------------------------
| MN LL Addr. | New AP LL Addr. | Old AP LL Addr. |
----------------------------------------------------
```

6. PROTOTYPE WIRELESS SERVICES

To demonstrate the practicability of DIRAC in implementing wireless and mobility aware services, we design and implement three protocols within the DIRAC router framework. All three protocols are wireless specific and they are: (1) a link-layer informed fast handover protocol, (2) a channel-adaptive FEC-based packet forwarding solution and (3) link-assisted policing. The above three services sample the rich wireless-adaptive protocols in the literature. They *span* both the control and data planes, and both uplink and downlink. Not only do they provide considerable performance gains, they also showcase the viability of these new wireless services within our router framework, which are otherwise not feasible in current router systems.

6.1 Link-Layer Informed Fast Handover

The fast handover service showcases how the control-plane element benefits from link-layer feedback. When a mobile user roams between subnets in a campus environment and uses Mobile IPv6 only [13], the handover latency can be significant and the transient packet loss non-negligible [18]. A fast handover protocol helps to reduce this latency and minimize loss through setting up a tunnel between two access routers (ARs) during handoff, so that packets in transit can be forwarded by the old AR to the mobile node (MN) via the new AR. Recent work [6, 19] proposes to use link-layer triggers to either predict or respond to a handover event. Our protocol adapts such earlier proposals [6] to the specific 802.11 link-layer technology. We do not claim much novelty of our design; we use it to showcase how such protocols can be easily implemented within DIRAC.

The main idea of the proposed solution is as follows: when a MN roams from one cell in a subnet to another cell in a different subnet, link-layer handover will happen first. This signals the earliest time the IP-layer fast handover may happen. We may simply use the link-layer handoff signal to trigger the IP-layer handover, and embed the IP-layer handover in the link-layer process. In the 802.11b network, we make use of the *ReAssociation* message of the link-layer handover process that precedes the network layer, and use it as the event that triggers the network-layer handoff service. Figure 7 shows the message exchange of the protocol. It involves 8 messages, similar to the recent IETF draft [6].

The protocol is implemented as a compound element in Click, i.e., *IP6FastHandover* in Figure 4. It consists of four elements in Figure 8. *IP6MobilityNewAR* implements the operations of the new access router for the mobile node. It

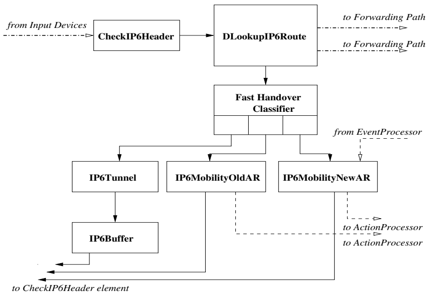

Figure 8: The compound *IP6 Fast Handover* element

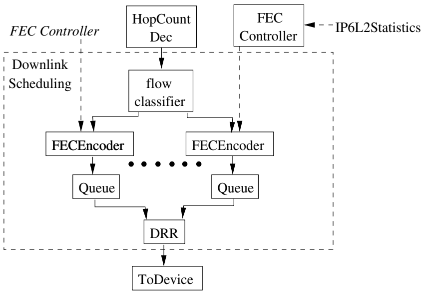

Figure 9: The compound element for the adaptive FEC

registers with *EventProcessor* in order to be notified upon the occurrence of a reassociation event. It also takes action employing *ActionProcessor* to allow link-layer association by the RA. *IP6MobilityOldAR* implements the operations of the old access router for the mobile node, while *IPTunnel* and *IP6Buffer* elements establish tunnels and temporarily buffer packets, respectively.

This protocol also requires implementation efforts on the mobile node part. We developed the client-side functionality and integrated it with a publicly-available distribution of Mobile IPv6 [20] for Linux. The protocol runs as part of the MIPv6 kernel module, and is enabled at runtime via a *sysctl()* call. This function also temporarily disables the native movement detection algorithm of MIPv6.

6.2 Channel-Adaptive FEC-Based Downlink Forwarding

Adaptive FEC-based forwarding seeks to address the wireless Head-of-Line (HoL) blocking problem. When the destination host of the HoL packet experiences channel error burst, the packet is retransmitted continually until the maximum retransmission count is reached. This blocks transmissions of subsequent in-queue packets destined to other hosts perceiving error-free channels. Its negative effect includes considerable transmission delay and jitter, and reduced throughput. Many recent wireless scheduling proposals solve the above problem in principle. However, this is typically achieved with per-packet information feedback, incurring considerable overhead. We thus devise a lightweight FEC-based solution.

The protocol works as follows: the RC requests an action from the RA to disable retransmissions, since they incur HoL blocking over the error-prone channel. However, in order to compensate for loss of reliability due to lack of retransmissions-based error recovery, FEC is used to recover losses. It is implemented at the Router Core, which adapts its encoded redundancy based on link-layer channel statistics provided by the RA. RAs do not have retransmission queues, and report link-layer statistics measured in frames lost due to errors. Lost frames are identified by the respective lost ACKs of the 802.11 protocol. The FEC-based forwarder implements Deficit Round Robin scheduling on a per-host basis, to avoid transmitting consecutive packets from an error-prone flow.

The chosen FEC algorithm [21] operates at the packet level. It takes two parameters (n, k), where k denotes the number of application packets, and n denotes the total number of packets (including the redundant $n-k$ packets) to be transmitted during an interval $[t, t+T]$. We adjust k and n based on the channel statistics, i.e., the frame loss[2] measured for each host and calculated as $frames\_loss = \frac{frames\_errors}{frames\_txmit}$, where $frames\_errors$ is the number of lost frames due to errors, and $frames\_txmit$ is the total number of transmitted frames. Then, k and n are adjusted periodically to satisfy the relation of $\frac{k}{n} = 1 - frames\_loss$. The receiving host is able to *reconstruct* the original k packets whenever it receives *any* k copies out of these n packets.

The above design is implemented in two Click elements shown in Figure 9: *FECController* and *FECEncoder*. *FECController* periodically queries the *IP6L2Statistics* element for channel statistics, and adjusts k and n of the *FECEncoder* element. *FECEncoder* encodes packets sent by *FlowClassifier*, and produces $(n-k)$ redundant packets. The packet header used for the adaptive FEC is shown in Figure 10. The 8-byte header is derived from the requirements for decoding at the receiver using the FEC algorithm of [21]. Ideally, this header should be implemented as a Destination Options Header in the IP. Our current implementation implements it after the UDP header of the packet to minimize development effort.

1 byte	1 byte	2 bytes
group_id	index	pkt_size
k	n	pkt_length

Figure 10: Header used for the adaptive FEC

6.3 Link-Layer Assisted Uplink Policing

This protocol seeks to police aggressive uplink flows via link-layer mechanisms. In a link-layer unaware scheme, the router *posterior* drops all the packets once they reach the router, but they have already consumed the wireless bandwidth. In the link-layer assisted design, the router uses the link-layer access control mechanism at the AP, i.e. 802.11 Deauthentication, to police the aggressive flow. This effectively denies such a flow *temporary* network access over the wireless channel. By appropriately choosing the period for access denial, it prevents the aggressive flow from stealing extra wireless link bandwidth.

The protocol has two building blocks: arbitrator and enforcer. The arbitrator decides which uplink flows are aggressive by checking a flow's actual transmission rate R_{actual} against its expected fair share R_{share}. The enforcer takes

[2] We also experimented with other link-layer metrics, e.g., channel quality expressed in dBm reported by the wireless card. However, our results show that there seems to be no quantifiable correlation between this value and the actual frame loss.

node (role)	*Hardware*
S (Source) (CN/SA)	Intel Pentium III 450MHz, 128MB Intel Ethernet Express 10/100
MN1, MN2, MN3 (Mobile Nodes)	Intel Pentium III 1.1GHz, 256MB Tulip & Netgear Fast Ethernet Cards
R1, R2 (Routers)	Intel Pentium III 900MHz 256MB, Intel Ethernet Express 10/100

Table 3: Testbed components used in the experiments.

two progressively severe shaping actions with increasing dropping intensity over time: RED-like dropping, and temporary denial of access. For aggressive, responsive flows (e.g. TCP friendly flows), RED-like dropping should be sufficient to shape the flow over the uplink wireless channel. For persistent non-responsive flows, temporary denial of access controls the damage. However, such flows are still granted long-term fair share over $[t, t+T]$. This is achieved by setting the access denial period t_{denial} as $t_{denial} = (1-R_{share}/R_{actual}) \cdot T$. The t_{denial} value is notified to the RA which informs the link layer to enforce it.

The protocol is implemented as a Click element. It invokes the 802.11 Deauthentication mechanism in the device driver to temporarily deny a flow's access.

7. SYSTEM EVALUATION

The evaluation of our "proof-of-concept" prototype seeks to answer two questions:

- How much overhead do the primitives of link-layer information introduce to the system?
- How beneficial is such an "informed" operation of the router for the wireless part of the network?

For the first question, we measure the overhead for processing *events*, *actions*, and *statistics*. For the second question, we evaluate the three wireless services of Section 6. The network components used in the testbed experiments are listed in Table 3. The specific configuration for each experiment is to be described in each subsection.

7.1 System Overhead

We first gauge how much processing overhead DIRAC incurs. We measure the consumed CPU time using the Pentium cycle counter [22], when running the code relevant to the processing of *events, actions*, and *statistics.* The results are then compared with the forwarding cost associated with delivering a packet from one interface of the RC to another. This provides a relative measure for the overhead of the control engine in DIRAC, and demonstrates the scalability of the approach in an enterprise environment, where there are a large number of APs connected to the same access router, all of which transmit link-layer information periodically.

The microbenchmarks for this experiment are collected on machine R1 of Table 3, which serves as the RC. We perform repetitive invocations of the functions that process the primitives of Section 4, and compute the average cost of each. The cost of the basic packet forwarding is defined as the amount of CPU time spent in the elements of Figure 4 of the forwarding path divided by the number of packets delivered. It is obtained by using host S of Table 3 as the source, host MN1 as the destination, and R1 as the router. The input load is 100,000 packets, generated at a relatively low rate in order not to overwhelm the hardware that runs the RC with interrupt handling.

Operation	*Time*(ns)
Action	498
Event	1712
Statistics report	32
Basic forwarding	1299

Table 4: Average CPU time cost for the primitives and basic forwarding (in nanoseconds).

Table 4 gives the experimental results. The processing overhead for each of the three primitives of actions, events and statistics is comparable to the forwarding cost. Actions and Statistics incur minor processing overhead, but events incur more. Specifically, for *statistics* processing, only $32ns$ are required by the *IP6L2Statistics* element to extract and store the values for a report from a RA. Our experiment also shows that additional $95ns$ are required to classify the packet as a report message that should be pushed to the *IP6L2Statistics* element for further processing.

The large difference between processing a statistics report and an event, is because the former has been highly optimized: only three memory-copy operations using pointer arithmetic are required to extract the values stored in the 10-byte report of channel quality statistics per host. However, the *EventProcessor* element still uses the highly convenient, but expensive string operations that the Click framework provides. It can definitely be optimized similarly; we leave it as future work. It should be noted, however, that *events*, as well as *actions* do not happen very frequently in practice, probably in the order of once every few hundreds of milliseconds at most. Therefore, the system overhead associated with them is not a primary concern.

To explore the scalability of the RC with the number of registered APs, we set up machine S as a source that transmits packets carrying dummy statistics. The UDP packets are generated directly from the kernel to avoid the expense of system calls. They are transmitted at several rates, to emulate various numbers of APs (from 1 to 200), each of which transmits one report every $50ms$. Each packet carries reports for 10 clients. The results are shown in Figure 11. From Figure 11, we note that the overhead for extracting and storing a large number of statistics from several clients is affordable. For example, the cost for processing 1000 packets per second from 50 APs is in the order of $140\mu secs$. It also scales linearly as more APs register with the RC. However, it should be noted that these results show the performance of the RC in a better light than it might be seen in a real network: packets are generated from a single source and all reports carry statistics for the same "clients." These two factors exploit the L1 data cache of the processor. Nevertheless, the results give an estimate about the overhead associated with processing statistics reports, a potential performance bottleneck.

To study the scalability of the RC as the number of mobile nodes associated with each AP increases, we use the setup of the previous experiment. Packets are transmitted at a high transmission rate of 2,000 packets per second, emulating the operation of 100 APs connected to the RC, each of which transmits a report every $50ms$. The results are plotted in Figure 12. From Figure 12 we observe that even when the number of mobile clients associated with a cell increases, the overhead is still minimal. For example, the cost of retrieving the channel statistics for 50 mobile nodes that are all carried

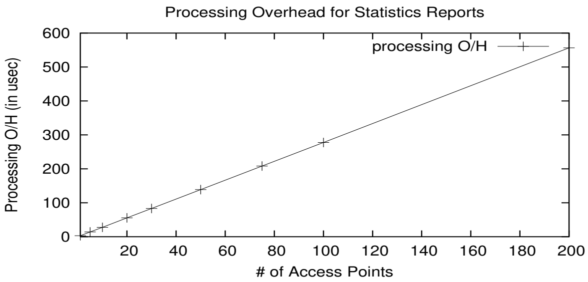

Figure 11: Overhead for processing reports from multiple access points (in microseconds)

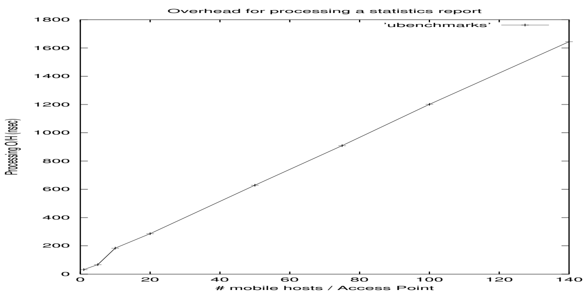

Figure 12: Overhead for processing a single statistics report (in nanoseconds)

in the same packet is only $629ns$, still 50% smaller than what is required for basic forwarding. As a final observation, we note that as the number of clients grows, the overhead grows somewhat sub-linearly. Since the same small piece of code is executed repeatedly, instruction cache hits (L1 I-cache of 16Kbytes) minimize the clock-cycle overhead.

7.2 Performance of Wireless Services

The ultimate goal of DIRAC as a wireless router system is to improve the router performance in wireless networks. This will be achieved through implementing wireless services described in Section 6, which address host mobility, improve packet forwarding over wireless links, and provide better sharing of the channel. Our measurements show that such services indeed provide considerable performance gains.

7.2.1 Host mobility

We use the testbed configuration of Figure 13 to evaluate the performance of the link-layer informed fast handover proposal of Section 6.1. A node roams from the vicinity of the cell served by access router R1, to the one served by R2. Node S acts both as a Home Agent (HA) for mobile host MN1 and as a Correspondent Node (CN) that transmits traffic to MN1.

A total number of 10 handoffs are performed, and different types of traffic (CBR, video over UDP, audio over TCP) are transmitted from one run to another. We use *tcpdump* to measure the time when the signals of the protocol are sent and received on the new access router (R2). Table 5 breaks down the delays of signaling messages involved in the fast handover protocol (see also Figure 7). The table shows that only 7.9ms –in addition to the inevitable link-layer switching delay from one AP to another, typically several hundred

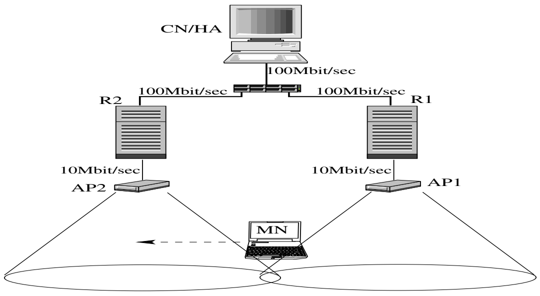

Figure 13: Experiment environment for the fast handover protocol evaluation

	mean	median	stdv
HAck - HI	4.2	3.6	1.5
ReAssocReply - ReAssocReq	7.9	7.4	1.8
PrRtAdv - ReAssocReply	18.6	18.8	3
FBU - PrRtAdv	8.1	6	6

Table 5: Latency involved in the link-layer informed fast handover protocol (in milliseconds).

milliseconds [23]– is required to establish the tunnel, which is created after the ReAssociation Reply message is sent by the new AR. Once the tunnel is set up, data packets for the MN arrive at the new AR via the tunnel with the old AR, thus minimizing transient packet loss.

From Table 5, it takes the mobile node 18.6ms to receive its new Care-of-Address and gateway (PrRtAdv message) after it starts receiving packets from the new point of attachment. This delay is large compared with the other table values. This is due to the fact that the PrRtAdv message is sent after the receipt of a notification message from the RA that completes transmission of the ReAssocReply message. However, since the Router Agent runs as a user-level program at the AP, its execution is deferred due to the execution of the driver code (also the code for the bridge), which runs in the kernel, has higher priority, and keeps the processor busy by forwarding data traffic.

Figure 14 plots the average end-to-end delay of UDP packets transmitted at a rate of 50 packets/sec with a packet size of 512 bytes, during the layer-3 handoff experiment. The particular version of our wireless cards (Orinoco Gold for the mobile node and Prism2 for the AP) gives a link-layer (L2) switching latency of around 500ms, which matches the value reported by another independent experiment in the literature [23]. The tunnel setup time of the L3 handover protocol is 8.2ms, after which the packets are transmitted over the tunnel for about 5 seconds (we artificially delayed the transmission of a Binding Update of the MIPv6 protocol [13], to observe the tunneling effect). Shown in the figure, for the duration of the tunnel, packets experience an increase of 5ms in their end-to-end delay, making the introduced overhead acceptable even for demanding real-time applications. As a last note, the latencies measured are very close to the ones reported by Koodli and Perkins in [10].

7.2.2 FEC-based downlink forwarding

We now demonstrate the Head-of-Line blocking problem of Section 6.2, and evaluate the FEC-based solution.

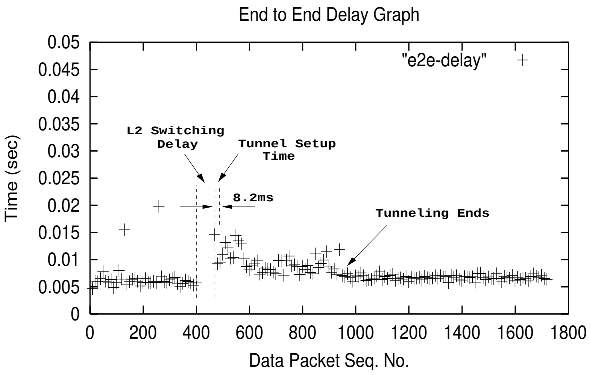

Figure 14: Average end-to-end delay of UDP packets during the fast handover experiment.

average quality	stdv	MN1	MN2	MN3
10.38	1.53	9903	9830	2647
12.38	1.46	9980	9940	7450
15.02	2.01	9980	9978	9581
20.17	1.54	9943	9903	9985
25.32	1.23	9925	9951	9984
35.53	1.97	9942	9936	9949
44.77	2.27	9928	9927	9954

Table 6: Packets received by each node, for various channel qualities of node MN3, using the FEC-based solution.

The testbed consists of a source machine (S), an access router (R1), and three mobile nodes (MN1, MN2, & MN3). The source transmits 10,000 packets, 1024 bytes each, to each mobile node at a rate of 65 packets/second/node. Two of the mobile nodes (MN1, and MN2) experience excellent channel quality. For the third one, we change its location by selecting seven spots so that its perceived channel quality varies from really bad to very good, one spot at a time, as shown in Tables 6 and 7. We compare the performance of the FEC-based solution against the default method for link-layer retransmissions, the default value of which is 8 in our cards. For each location and method, we run 3 rounds of experiments interleaving the two methods, in order to minimize fluctuations in channel quality due to interferences. Moreover, to provide fair comparisons, we lock the receiving rates for all mobile nodes to 11Mbps, by disabling the multirate support of the driver in the AP card. The link-layer statistics sampling interval at an AP is 100ms.

Tables 6 and 7 show the number of packets received by each node, out of the 10,000 packets sent to each one of them, for various channel qualities experienced by MN3. From the tables, we make the following observations:

First, the HoL blocking problem and its solution by the FEC-based approach are demonstrated from the results of the first two rows of both tables. With the link-layer retransmission method, the throughput of MN1 and MN2 has coupling effect with MN3. When MN3 experiences bad channel and its throughput suffers, so do MN1 and MN2. However, in the FEC-method, this negative effect does not show up any more. Even though MN3 still suffers from low throughput, MN1 and MN2 enjoy very high throughput. This exactly shows the impact of HoL blocking. In the retransmission method, excessive retransmissions for MN3 cause buffer overflows at the AP. This buffer drop also happens for MN1 and MN2, since all three share the buffer at the AP. However, the FEC-based approach, which also produces lots of redundant packets for MN3, causes buffer overflow only for the queue of MN3 at the RC. This early drop at RC, rather than AP, leads to isolation for the other two flows; the FEC-based approach has an inherent rate-control mechanism. One could possibly implement a rate-control mechanism to ensure early dropping for the flow experiencing bad channel conditions and solve the HoL blocking problem, while still using the retransmissions-based method. However, the implementation on the access point seems much more involved, due to the modifications required in the driver.

average quality	stdv	MN1	MN2	MN3
9.47	2.17	4808	4807	3038
12.21	1.81	7559	7554	6558
14.81	1.76	9999	9999	9822
20.44	1.98	9999	9999	9988
25.38	1.15	9999	9999	9990
35.11	2.29	9999	9999	9996
45.35	2.42	9998	9999	9998

Table 7: Packets received by each node, for various channel qualities of node MN3, using the default method of link-layer retransmissions.

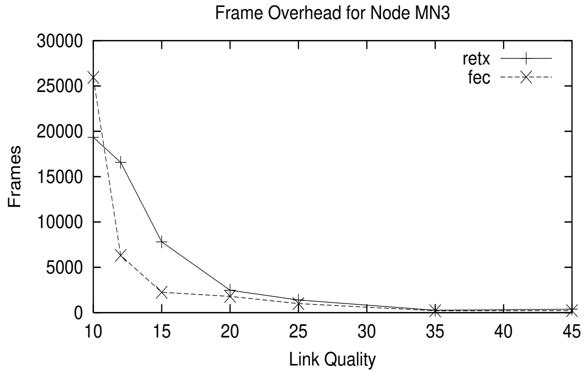

Figure 15: FEC overhead

The second observation has more to do with the wireless channel itself: the more the quality metric (measured in average quality and standard deviation in the tables) approaches the value of 10, which seems to be the threshold at which the wireless card of the mobile node starts scanning for another AP to initiate handoffs, the sharper the drop is in the number of packets received. At higher values, the channel behaves in a more predictable way.

Figure 15 plots the communication overhead using these two methods. The overhead is defined as any redundant copies transmitted over the air, whether it is due to retransmissions or due to FEC-generated redundant packets. The figure shows that in many scenarios, FEC may even outperform the retransmission-based method; this is a bit counter-intuitive. The main reason is that channel error happens in bursts. For bursty errors, the retransmissions method leads to consecutive packet losses. The FEC method may schedule packets from other flows, thus avoiding back-to-back transmissions in the presence of error bursts.

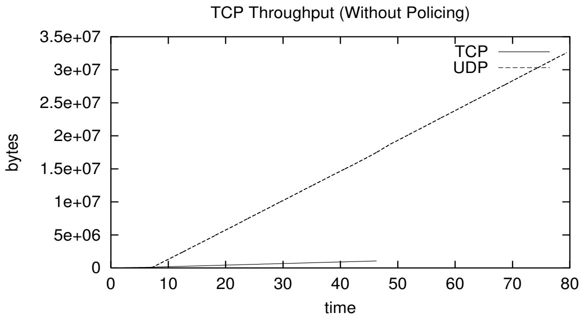

Figure 16: Throughput without policing

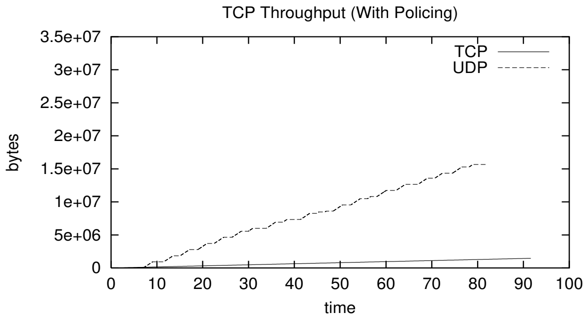

Figure 17: Throughput with policing

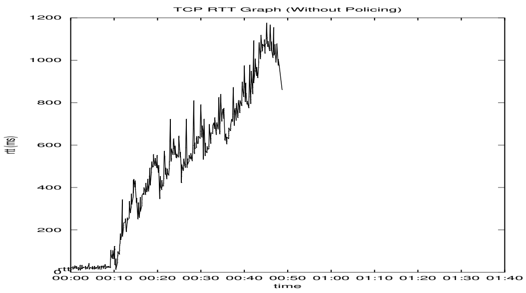

Figure 18: RTT without policing

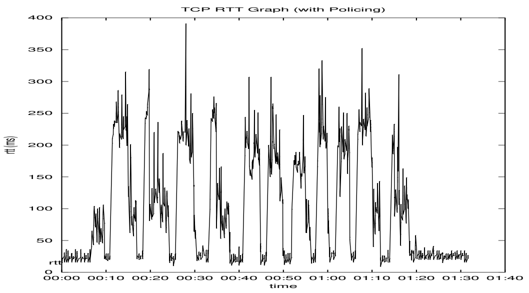

Figure 19: RTT with policing

7.2.3 Policing

We evaluate the effectiveness of policing with a testbed consisted of two mobile nodes MN1 and MN2 competing for the wireless channel. MN1 is running Icecast [24], an MP3 streaming server that uses the TCP protocol to deliver audio streams at 128kps. MN2 is an aggressive UDP source that generates 3.6Mbps of traffic, enough to consume the most of the channel by itself.

Figure 16 shows that without a policing mechanism, the aggressive MN2 occupies a significant portion of the wireless channel and leaves little operational bandwidth for MN1. From the figure, we can see that the TCP operated by MN1 is disrupted midway since it can no longer operate throughout the experiment. Figure 18 reflects the increase of the TCP round-trip-time for MN1.

In Figure 17 with policing, MN1 is protected from MN2's destructive behavior. Whenever the RC deems the MN1 being overly aggressive, it will instruct the AP to penalize MN1 by temporarily restricting its access to the AP. Thus the benevolent MN2 can still operate at its full capacity. The figure shows that the throughput of MN2 is constantly being limited by the RC and the MN1 is able to run to the completion of our experiment without being shut down. Figure 19 is the respective TCP round-trip-time graph.

In summary, DIRAC provides primitives for interacting with the wireless link-layer without introducing much overhead to the system. It scales well to the number of mobile nodes, consuming processing power less than 2.5% of the cost for a standard packet forwarding. It takes less than $1\mu sec$ to process channel states for 50 clients per AP. It can also support a large number of APs; less than $140\mu secs$ are required to process statistics reports from 50 APs, each of which transmits 20 reports/sec. Through its API, a number of network services critical to the wireless environment can be readily deployed. The fast handover service establishes a tunnel for packet forwarding in less than $10ms$. FEC-based forwarding solves the wireless Head-of-Line blocking problem with overhead comparable with the default retransmission based solution. The policing service protects flows from aggressive users.

8. DISCUSSIONS AND LESSONS LEARNED

8.1 Discussions

Scalability Wireless routers mostly serve as *edge routers* and do not sit within the Internet backbone. Moreover, wireless channel capacity and constrained communication range both limit the number of flows supported by a wireless router to be orders of magnitude smaller than the wired scenario. However, each router may have to serve hundreds of APs.

Applicability in outdoor wireless WANs The link state used in the router system, such as link quality of each host, is rather generic and available also in wide-area cellular networks. The link-layer mechanisms and management messages, e.g. the layer-2 handover signal and the access control mechanism, are also available in such networks. Both aspects point out that the architectural components still apply and the system can be adapted to such WWAN networks with slight modifications of the router agent.

Changes on the mobile host To use wireless services enabled by DIRAC, the mobile host may also need to update its software. For example, the current implementation requires the host to install modules to support fast handover, mobile IPv6, and adaptive FEC. However, we stress that the channel state information–frame loss statistics–is retrieved at the AP, not from each mobile host. The device driver at the AP computes the loss statistics based on the link-layer ACK mechanism: a packet is considered lost if the DATA frame is sent but ACK is not received.

8.2 Experiences and Lessons Learned

The first learned lesson is that, as in other systems areas, it is important to "make the common case fast." This is certainly true in the router design context, particularly on the data path. In our system, processing channel statistics is an important common case. Therefore, it is critical to minimize the processing of link-layer statistics, in order to allow the router to scale to a large number of hosts and APs.

The second issue involves the OS scheduling of computations on a device that performs forwarding. Naturally, a higher priority is given to the basic forwarding functionality, while other control-plane operations receive a smaller fraction of the processing capacity. This also applies to the RA on APs: Since the bridge performs a limited-scope L2 forwarding of Ethernet frames (from the wired interface to the wireless and backwards), the RA and user-level implementation of the 802.11 daemon have to be scheduled at regular intervals, in order to execute the management functionality. This can be done either by having the bridge giving up the CPU voluntarily, in order to allow execution of the router agent, or by implementing the latter inside the kernel.

The last issue is related to the collection of statistics that characterize channel quality on a per-host basis. Initially, we implemented a mechanism so that mobile nodes send periodic reports back to the AP, regarding the perceived channel quality. These reports used Signal-to-Noise ratio measured in dBm. However, the estimation of frame loss from this metric was not very accurate. For this reason, we went ahead and modified the driver of the AP card to obtain the frame loss statistics of Section 5. A more elaborate set of statistics can be collected via the *CommTallies* [16] facility of the wireless card, but unfortunately they are not available on a per-host basis.

9. RELATED WORK

Motivated by enhancing routers with new capabilities such as assorted filters, firewalls, and traffic prioritization, router design for the wired Internet has been an active research area recently. Most of such routers are software based and seek to support easy customization of router functionality. Examples of such systems include Scout [25] and its extension with network processors [1], Click [8], Router Plugins [26] and XORP [27]. Although all of these proposals provide extensible and general frameworks for router development, they lack specific support for working in wireless networks. A distributed router design appears in [28]. However, the focus there is to achieve high throughput through a cluster of PCs. In [29], which can also be viewed as a distributed design over the hierarchy of network processors, the focus is on supporting active networks.

There are home wireless router systems available in the market, including the 3Com home wireless router, Intel AnyPoint Home Network, and Lucent Orinoco RG-1000 Residential Gateway, to name a few. The main focus of such systems is to bridge the wireless channel with the wired Internet. The claimed new features are mainly on security such as integrated firewall. These systems do not take a distributed architecture. The MIT personal router [30] is concerned with the automatic selection of providers of wireless access, based on dynamic modeling of the user and knowledge of a market of wireless services. DIRAC serves as network edge router and improves the performance of forwarding engine over the wireless channel and support fast handover.

Other commercial solutions for the enterprise environment attempt to address similar issues but in a narrow scope. The Vernier Networks 6500 [31] system focuses mostly on security and management. The goal is to secure a wireless network by enabling security protocols and enforcing access rights across multiple network domains. The Lucent SpringTide 7000 Wireless IP Service [32] provides several mobile IP services mainly through tunneling. The Nomadix Service Engine Software Modules [33] use techniques based on ARP to identify new users and address user mobility. No data-plane forwarding issues are addressed in these systems. In Aruba Network's Aruba 5000 and Aruba 50 systems [34], the proprietary APs transmit link-layer information feedback to the WLAN switch. However, it still falls into the category of intelligent AP systems. The main goal is to enhance security and roaming functions. The design is still mainly at the link layer and stays on the control plane.

Many components in the router system have been studied in the literature. Wireless scheduling has been an active research area in recent years [3][4][5]. Several fast handover solutions [6][18] have been proposed, and system solutions for mobility support have been devised and implemented [35][36]. Our distributed router architecture borrows freely from the literature, but is not tied to a particular protocol. It rather provides all those primitives that are necessary for the implementation and deployment of these protocols.

Cross-layer interactions have been a popular design guideline in wireless networking. The Mobiware toolkit [37] spans several layers of the protocol stack, from programmable MAC, to transport and adaptive applications. We take a more controlled approach to cross-layer interactions in DIRAC, to reduce overhead, achieve better scalability and extensibility.

10. CONCLUSION

Router is expected to be a critical architectural component in the emerging packet-switched cellular networks, such as campus or enterprise wireless network based on 802.11 technology, and metropolitan wireless networks. Conventional routers do not handle mobility and wireless link issues. The solution principle, well documented in the literature, is to take adaptive actions based on channel conditions and mobility patterns.

This paper describes DIRAC, a wireless router system. The goal is to design and implement a systems framework that enables wireless protocols and provides router services for wireless data networks. DIRAC's unique approach is to let the router core interact, in a controlled manner, with the wireless link layer, which is in the best position to know the network dynamics. The interaction is twofold: the router core takes feedback input from the link layer, and dictates actions to the link layer. This leads to renovated designs in both data and control planes of a router and facilitates implementation and deployment of new wireless services.

To this end, we implement a distributed router architecture, in which the router core interacts with each router agent at each AP. We also implement three prototype wireless services, spanning both control and data planes, and covering both the uplink and downlink forwarding path on the data plane. Together, they showcase how a wide spectrum of wireless protocols can be implemented and evaluated within DIRAC. DIRAC is built using off-the-shelf hardware, including a PC and several APs.

Ongoing work seeks to support firewalls for roaming users, implement a inter-AP, coordinated resource management protocol to support VoIP, to add energy efficiency feature via power-saving mode for mobile hosts, to explore performance optimization techniques in DIRAC, and to gain more experiences through a larger testbed deployment.

11. ACKNOWLEDGMENTS

We greatly appreciate the insightful comments by our shepherd, Dr. Ramesh Rao and the constructive critiques by the anonymous reviewers. We also thank Dr. Lixia Zhang for helpful comments on early drafts of the paper.

12. REFERENCES

[1] T. Spalink, S. Karlin, L. Peterson, and Y. Gottlieb, "Building a robust software-based router using network processors," *SOSP'01*, October 2001.

[2] Y. Gottlieb and L. Peterson, "A comparative study of extensible routers," *OpenArch'02*, June 2002.

[3] S. Lu, V. Bharghavan, R. Srikant, "Fair queueing in wireless packet networks", *SIGCOMM'97*, 1997.

[4] T. Nandagopal, S. Lu, and V. Bharghavan, "A Unified Architecture for the Design and Evaluation of Wireless Fair Queueing Algorithms," *MOBICOM'99*, August 1999.

[5] X. Liu, E.K.P. Chong, And N. B. Shroff, "Transmission Scheduling for Efficient Wireless Network Utilization," *INFOCOM'01*, April 2000.

[6] Fast Handovers for Mobile IPv6, *draft-ietf-mobileip-fast-mipv6-05.txt*, 2002.

[7] J. Ahn, and J. Heidemann, "An Adaptive FEC Algorithm for Mobile Wireless Networks," *Technical Report ISI-TR-555*, USC/ISI, March 2002.

[8] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M.F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems*, August 2000.

[9] Open AP Platform *http://opensource.instant802.com/*.

[10] R. Koodli and C. E. Perkins "Fast Handovers and Context Transfers in Mobile Networks," *ACM CCR*, 31(5), October 2001.

[11] D. Kotz, and K. Essien, "Analysis of a Campus-wide Wireless Network," *MOBICOM'02*, September 2002.

[12] Juniper Research. *http://www.juniperresearch.com*.

[13] D. Johnson, C. Perkins, J. Arkko, "Mobility support in IPv6," *http://www.ietf.org/internet-drafts/draft-ietf-mobileip-ipv6-18.txt*.

[14] T. Narten, E. Nordmark, and W. Simpson "Neighbor Discovery for IP Version 6 (IPv6)," IETF RFC 2461, December 1998.

[15] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, K. R. Walker, "Agile Application -Aware Adaptation for Mobility," *SOSP'97*, 1997.

[16] Intersil Prism2 driver. *http://hostap.epitest.fi/*.

[17] A. Rubini, and J. Corbet, *Linux Device Drivers, 2nd Edition*, O'REILY, ISBN 0-596-00008-1.

[18] H. Yokota, A. Idoue, T. Hasegawa, and T. Kato, "Link layer assisted mobile IP fast handoff method over wireless LAN networks," *MOBICOM'02*, 2002.

[19] P. McCann "Mobile IPv6 Fast Handovers for 802.11 Networks," *http://www.ietf.org/internet-drafts/draft-mccann-mobileip-80211fh-01.txt*.

[20] Mobile IPv6 for Linux. *http://www.mipl.mediapoli.com/*.

[21] L. Rizzo, "Effective erasure codes for reliable computer communication protocols," *ACM CCR*, pages 24-36, 1997.

[22] "The IA-32 Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide," *http://developer.intel.com/design/pentium4/ manuals/245472.htm*.

[23] A. Mishra, M. Shin and W. Arbaugh "An Empirical Analysis of the IEEE 802.11 MAC Layer Handoff Process," Tech Report, UMIACS-TR-2002-75.

[24] Icecast Streaming Server, *http://www.icecast.org/*

[25] D. Mosberger and L. Peterson, "Making paths explicit in the Scout operating system," *OSDI'96*, 1996.

[26] D. Decasper, Z. Dittia, G. Parulkar, and B. Platter, "Router plugins: A software architecture for next generation routers," *IEEE/ACM Trans. on Networking*, February 2000.

[27] XORP: Extensible open router platform. *http://www.xorp.org/*.

[28] P. Pradhan and T. Chiueh, "A Cluster-based, Scalable Edge Router Architecture," Technical Report, *http://www.ecsl.cs.sunysb.edu/ prashant/papers/ design.ps.gz*.

[29] N. Shalaby, L. Peterson, et al. "Extensible Routers for Active Networks," Tech Report, *http://www.cs.princeton.edu/nsg/papers/dance.pdf*

[30] P. Faratin, J. Wroclawski, G. Lee, and S. Parsons, "The Personal Router: An Agent for Wireless Access," *AAAI'02*, July 2002.

[31] Vernier Networks System 6500. *http://www.verniernetworks.com/AMCS6500.html*.

[32] SpringTide 7000 Wireless IP Service Switch Router. *http://www.lucent.com/livelink/0900940380004ac9 _Brochure_datasheet.pdf*.

[33] Nomadix Service Engine. *http://www.nomadix.com /downloads/products/ NSE_Data_Sheet.pdf*.

[34] Aruba 5000. *http://www.arubanetworks.com/ products/5000/*.

[35] A. Miu and P. Bahl, "Dynamic host configuration for managing mobility between public and private networks," *Usenix Internet Technical Symposium*, March 2001.

[36] M. E. Kounavis, A. T. Campbell, et al, "Design, Implementation and Evaluation of Programmable Handoff in Mobile Networks," *MoMuc 2000*.

[37] O. Angin, A. Campbell, et al "The Mobiware Toolkit: Programmable Support for Adaptive Mobile Networking," *IEEE Personal Communications Magazine*, August 1998.