

# The Case for Intentional Networking

Jason Flinn<sup>1</sup>, T. J. Giuli<sup>2</sup>, Brett Higgins<sup>1</sup>, Brian Noble<sup>1</sup>, Azarias Reda<sup>1</sup>, and David Watson<sup>2</sup>

<sup>1</sup>Computer Science and Engineering  
University of Michigan

<sup>2</sup>Research and Innovation Center  
Ford Motor Company

## 1. INTRODUCTION

Wireless infrastructures are increasingly diverse, complex, and difficult to manage. Those who restrict themselves to homogeneous, managed campus or corporate networks are a vanishing breed. In the wild, users are confronted with many overlapping infrastructures with a broad variety of strengths and weaknesses. Such diversity of infrastructure is both a challenge and an opportunity. The challenge lies in presenting the alternatives to applications and users in a way that provides the best possible utility to both. However, by managing these many alternatives, we can provide significant benefits, exploiting multiple networks concurrently and planning future transmissions intelligently.

To this end, we are developing *Intentional Networking*—a set of interfaces and mechanisms that allow applications, users, and the operating system to proactively manage current and expected future connectivity. We do this through extensions to the networking API. Applications can label sockets or individual transmissions with a *label*, a qualitative statement about the flow. The operating system can then best match each flow with network capabilities. In some cases this requires a re-ordering of the application's send order; our API offers both blocking and event-based interfaces to allow this reordering.

Exposing multiple network interfaces that come and go—even under a single, virtualized contact point—presents a surprising number of challenges for “regular” applications that typically assume a single, long-lived interface. In particular, some services are not prepared to support clients with multiple points-of-presence, nor re-ordering across them. After outlining our interface design, we present a taxonomy of the issues that existing applications might face in adapting this new model.

Some applications do fit our model quite closely—particularly those designed with high-delay networks or volatile connection points. Such applications may be transparently adapted to our framework, but this requires inference of labels. This might be done through iterative adaptation using APIs that applications export, or possibly through stack introspection

and flow analysis.

## 2. DIVERSITY AND MOBILITY

As the wireless market continues to grow, two trends have become clear. First, there is a growing diversity in connectivity options. Second, users have come to expect and demand true mobility, rather than simply nomadicity.

There are several factors contributing to connection diversity. The tension between power, coverage, and bandwidth [16] has given rise to a variety of overlay networks—broad-area, low-rate coverage augmented with small pockets of high-speed connectivity where usage density warrants such deployments.

Such overlay networks are only one source of connection diversity—within a single layer, there is often significant variation in capacity and capabilities. For example, consider WiFi hotspot deployments. These access points are under decentralized control, are managed by a varied set of residents and businesses, and have different back-end wired connections to the broader network. Many APs reject or restrict foreign users in a variety of ways, and “front-side” wireless bandwidth is often not the bottleneck. Since there is no common administrative control, there is also no centralized database to guide users' selection policies in favor of the APs providing the best service. While many searchable databases of “wardriving” maps exist, these maps become outdated quickly and provide no information about access points apart from the basic information broadcast in the beacon signal.

At the same time, users increasingly demand connectivity while on the move. The growth in converged devices for constant connectivity has been unprecedented, and industries from health care [6] to automotive [13] are increasingly turning to constant connectivity to increase effectiveness and add value. These users expect near-constant connectivity, no matter where or when. This is markedly different from the nomadic, laptop model—where users move infrequently, and compute only when in one place.

## 3. ACTIVE CONNECTION MANAGEMENT

Rather than leave each application to its own devices, we propose a new model of network connectivity management at the operating system level. This *connection manager* is comprised of two main components. First, the **virtual link layer** resides in the kernel, between the existing network and link layers of the protocol stack. This component uses the physical network interfaces present on the device to connect simultaneously to all usable access points in the current

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

HotMobile '09 Santa Cruz, CA, USA

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

vicinity. The complexity of using multiple interfaces is hidden behind a single *unified network interface* that is the one network device visible to applications and users. All data flows are routed through this interface, which assigns each to the virtual network interface best matching the needs of the flow. Applications that are aware of this interface use labels to indicate needs of data flows with regard to interactivity, foreground/background priority, and bandwidth needs. Labels can be passed by setting socket options, or as an optional argument to `send`.

Second, the **connection scout** is a privileged, user-space process with a two-fold purpose. Its first task is to discover and evaluate new network access points, similar to the AP selection process described earlier. As it finds a new network connectivity option, it informs the virtual link layer so that it might be utilized by data flows. The connection scout informs the virtual link layer of the connection quality test results for each new access point, so that the virtual link layer can best match the needs of data flows with the capabilities of network interfaces. The second task of the connection scout is to maintain the device-centric mobility model. Based on the different WiFi access points, GSM and WiMax towers, and other fixed beacons the connection scout detects as it scans for new connectivity, it builds a Markov model that seeks to predict future device movement based on past behavior and the current location. Both applications and the kernel query this “oracle” and are free to apply the resultant prediction in application-specific ways.

Figure 1 gives a conceptual overview of how our *virtual link layer* functions. In the figure, our device has three different wireless radios: IEEE 802.11, Bluetooth, and WiMax. The virtual link layer creates a virtual network device inside the kernel for each available physical AP. In our example, there are three 802.11 APs present, two Bluetooth devices, and one WiMax broadcast tower, for a total of six virtual network devices.

In contrast, current kernels export one interface for each physical device. This forces a static association between interfaces and access points which cannot allow two simultaneous flows to use different access points with different characteristics for open ports, latency, bandwidth, etc. Further, current kernels use static routing tables to assign all flows with the same destination address to a single interface. Our virtual link layer, however, can assign multiple flows from the same application and the same destination address to different virtual devices, which causes them to use different physical devices or APs. This complexity is hidden from users and applications, with the exception that applications aware of our API can label some or all of their flows to empower the kernel to make better assignments.

When an application creates a new data flow (i.e., a new socket), regardless of which connectivity sources the user’s device has access to, the socket is bound to the single, unified network interface. Based on the qualities of service that are important to the flow, the flow is assigned to one of the virtual network interfaces. When data is received by one of the virtual network interfaces, it is forwarded up to the unified network interface. Many applications see this as a single interface regardless of whether it arrived via one of many WiFi APs, Bluetooth, WiMax, or some other link technology. Some applications—those with an internal notion of *connection*—require notice when interfaces change.

Different data flows require different types of network con-

nections. Some, such as `ssh` sessions, are interactive but have fairly modest bandwidth requirements. Others, such as peer-to-peer file transfers, may have large bandwidth needs but need not be completed in the foreground. Certain flows are sufficiently important that they should be serviced via links with high monetary cost if necessary. If the kernel was aware of such requirements, based on the AP property information determined by the connection scout it could assign data flows to the most appropriate of all available interfaces. Such information is most valuable in situations where many data flows are competing for limited network resources. Data flows that are high-priority or require a robust connection (such as VoIP calls) could still be serviced when connectivity is scarce, and less important flows would be deferred until the mobile device moved to a new area with a surplus of wireless bandwidth. This priority information would also be useful when deciding if a data flow is important enough to transmit over a costly GPRS or other pay link.

Unfortunately, applications, not the kernel, understand the semantics of data flows. The kernel is aware of objects such as sockets, but has no direct information about what the data passing through the socket is attempting to accomplish. To support this, our system pierces the abstraction barrier between applications and the kernel. When creating a new socket, applications transmit some additional information—declarative hints—to the kernel.

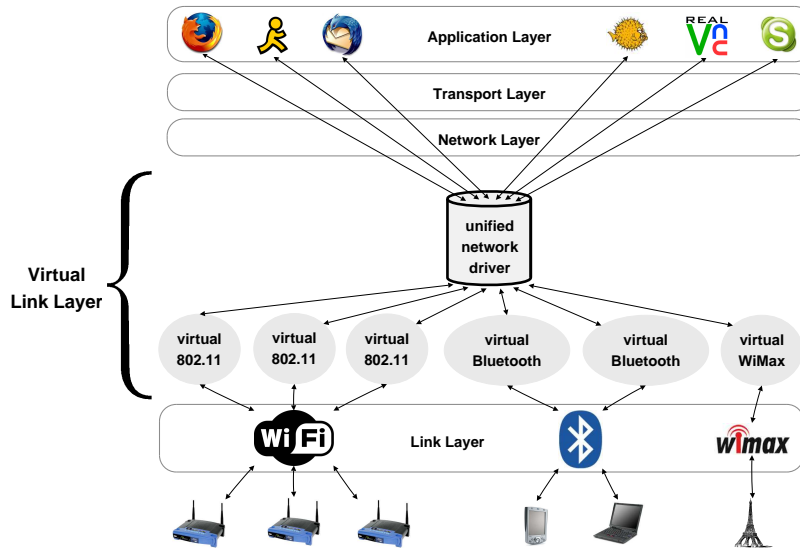
One might think to make these hints very detailed. For example, applications could be required to predict the average, minimum and maximum bandwidth and latency requirements of a socket. However, we believe this would impose an unreasonable burden on application programmers.

Instead, we define a simple interface that is expressive enough to provide the kernel with enough information while minimizing application burden. Applications communicate the following three Boolean values to the kernel on a per-socket basis:

- **Interactivity:** Is this flow part of an interactive task that should be completed as soon as possible to minimize wait time?
- **Bulk Transfer:** Is this flow a bulk transfer that requires substantial network bandwidth to complete?
- **High Value:** Is this flow sufficiently important that it should be serviced via a pay-per-use link (such as GPRS, or a commercial WiFi AP) if that is all that is available?

Once the quality of connectivity offered by access points and the connectivity needs of data flows are both known, the remaining task is to assign flows to the best interface. In current operating systems, when multiple network interfaces exist data flows are assigned to an interface based on network layer routing tables. For example, in IP networks each interface has an associated IP address and subnet mask. Depending on the destination IP address in each outbound packet’s header, and the subnet masks of the available network interfaces, the network layer routes the outbound packet to an interface. This model works fine when multiple network interfaces partition the IP address space among themselves for some purpose, such as when different interfaces are on different physical LANs.

We propose a very different paradigm in which network interfaces and APs are chosen based on declarative hints



**Figure 1: Virtual link layer architecture.** The bottom layer (link layer) consists of the device drivers that interface with the physical network interfaces of the device—one driver per radio type. Conceptually, we add a *virtual network driver* for each available AP for each radio type. In the figure, there are three virtual 802.11 drivers because three WiFi APs are present. The complexity of all these virtual drivers is hidden by a single, unified network driver that is the only network device visible to applications. In our actual implementation, one driver handles the entire virtual link layer to avoid overhead of message passing between multiple device drivers.

rather than network addresses. Our unified network interface has the responsibility of deciding to which of the available virtual network interfaces the flow should be assigned. When outbound data arrives at the unified interface, the module decides which physical network should handle traffic, based on declared or inferred flow characteristics.

There are times when the only available networks are ill-suited for the data at hand. For example, a large opportunistic transfer should not be sent over a low-bandwidth high-cost link. One could return an error in such cases, but this places the burden of when to retry to the application, rather than the connection scout. Instead, we offer two additional mechanisms in our interface. The first blocks the message until the network is available, with an optional timeout. The second returns immediately, but takes a closure as an additional argument. This closure is invoked when a suitable network is available. Closures may be canceled if so desired. We expect the blocking form to be more useful to threaded applications, while the closure form will better fit event-based applications.

These API features are extensions to the normal socket interface. Legacy applications can continue to use this interface, depending on the connection manager to map the application as a whole to the best available network. Those applications that are aware of the extensions can use them to benefit fully from the many alternatives available.

#### 4. APPLICATION TAXONOMY

This model of network management is more appropriate for some application styles than others. At one extreme, connection-oriented applications with no concurrency have very limited opportunities to take advantage of diversity and

prediction. However, applications that are able to be more flexible offer more opportunities. In considering how best to adapt an application to this model, we consider several dimensions.

**Homogeneous vs. heterogeneous traffic:** Many applications have a single set of labels for all traffic. Such applications can be handled transparently—their flows are mapped to the best fit at any given time. Others issue traffic with different needs over time, and they often require some application involvement beyond simply providing labels, as the rest of this section shows.

**Message passing vs. request-response:** The simplest applications to adapt are those that rely strictly upon message passing, without requiring responses for progress. Such applications can freely send across different physical interfaces with minimal consideration given to application semantics. Examples of such applications include opportunistic, gossip-based protocols, and those crafted to fit well with delay-tolerant networks [11].

**Strictly vs. loosely ordered:** Some applications—particularly those with connection-oriented semantics—assume that messages cannot be reordered by lower layers. Often, this assumption is unnecessary, but breaking it requires refactoring applications in several ways. In particular, such applications must declare the boundaries across which reordering is possible, but within which it is not. With datagram-oriented applications, individual calls to **send** implicitly identify these boundaries, but connection-based applications must explicitly mark them in some way. Finally, some applications cannot tolerate re-orderings within a single connection at all—such applications can generally use a single network at a time, switching among them as conditions change.

**Connection breadth and depth:** Connection-oriented applications can be further broken down by the *breadth* of the connections they expect or support. At the extreme, some applications expect only one connection from a principal—even one with access to more than one machine. For a time, the University of Michigan’s VPN was configured to allow only one connection from any machine for each authenticated user. Some applications assume a single connection between any particular client machine/server pair. Supporting such scenarios requires “re-connection” each time the connection manager changes endpoints. This may be as simple as a TCP `connect`, or it may require an application-level action such as authentication. In either case, the closure mechanism for deferred sends is overloaded to provide the necessary action. In the worst case, such reconnection is intrusive to the user; such applications are poor fits for our model.

## 5. INFERRING LABELS

Some existing applications meet many, if not all, of the constraints identified in the prior section. We expect to explore a variety of techniques to support such legacy applications. They include external adaptation and causal analysis combined with stack introspection to learn the communication patterns of applications.

The first technique is called iterative adaptation [8]. The insight of this approach is that many applications provide APIs that allow other services some degree of control over the application’s behavior and actions. These APIs have been used for adaptation in the resource-constrained environment of mobile data access without requiring any source code modifications. We expect to be able to apply the same approach; by intercepting user actions and correlating that to network usage, we believe it possible to identify on-demand activity, and may even be able to classify opportunistic behavior by observing UI updates that do (or, importantly, do not) happen together with I/O activity.

The second technique is more involved, but requires no exogenous application support. It combines stack introspection techniques from the security community [33] with causal analysis techniques recently used to provide high-performance file systems that provide strong persistence guarantees [24]. This scheme tracks user and UI behavior through the operating system, identifying the set of inputs that can possibly have influenced a set of outputs.

Of course, this set is possibly too large, because it tracks any relationships that *might* have been causal. However, there are some techniques that we can use to prune the set. For example, by observing many executions of similar code paths, we can eliminate candidate causal events that only happen some of the time [17]. Likewise one could apply taint checking to track causality within a process [22]. Unfortunately, such taint checking is extremely expensive, but progress has been made in paralleling the task [23].

It is important to be conservative in any of these schemes. However, it is also possible to apply them offline, or in controlled environments, or even post-hoc given user behavior traces. This allows the application of heavy-weight mechanisms such as taint analysis offline, and using the results of that analysis in online stack introspection.

## 6. RELATED WORK

The push toward ubiquitous computing makes automatic service discovery in new environments more important than ever [29]. Existing work, however, has focused more on enabling application-level services [7, 12, 31] than choosing and managing a diverse set of network connections.

Several systems seek to allow clients of one wireless service provider to access foreign wireless hotspots when roaming [3, 10, 19, 28] or between public and private networks [20]. Our work is complementary, since users must find and associate to an access point before negotiating such roaming agreements. This service discovery is similarly critical for grassroots wireless collective initiatives [2, 25, 30].

Marmasse [18] argues, as we do, in favor of a user-centric mobility model. Her *comMotion* system is concerned chiefly with tracking users’ movement through various semantically meaningful locations, such as “home” or “work”. We, on the other hand, focus on lower-level waypoints—namely, wireless connectivity points. The semantic concept of user-defined locations could easily be layered atop such low-level information, however.

MultiNet—now known as Virtual WiFi— [5] virtualizes a device’s wireless interface, fooling applications into believing the device is connected simultaneously to different APs on different channels. This is a step in the right direction, because devices can now exploit all available connectivity in their vicinity. Unfortunately, the complexity of assigning data flows to interfaces is still present. All available APs are presented to higher layers of the network stack as if the device had a wireless radio for each AP that is present. We propose implementing similar functionality, to both allow searching for new APs in the background and allow concurrent utilization of all wireless networks the device can see at a given location. Applications and users will only be aware, however, of a single unchanging network interface. Unlike Virtual WiFi, the kernel will be responsible for assigning data flows to a certain virtual interface. FatVAP [15] presents a similar infrastructure, but operates only within a single layer of an overlay network, and is concerned only with maximizing overall throughput, without concern for other application-level preferences.

Contact Networking [4] hides the differences between local and remote communication from users. All communication appears to be local—like a direct Bluetooth connection between two devices—even if infrastructure such as the Internet is actually involved. Like us, the authors recognize that mobile devices typically have several, heterogeneous wireless radios at their disposal. Contact Networking is also conscious of the properties of different link layers. Their primary focus, however, is on neighbor discovery, name resolution, and (ultimately) the preservation of application-level sessions in the face of user mobility. Our work does find common ground with the idea that all network connectivity options are not equivalent, and the operating system should dynamically assign data flows to the most appropriate link.

Zhao, Castelluccia, and Baker’s work [34] attacks similar problems as Contact Networking. Their work lies firmly within the framework of Mobile IP [27] as well. The user’s Home Agent is required to arbitrate the routing of various data flows. Furthermore, applications must explicitly bind a data flow to a specific interface through their `SO_BINDTODEVICE` socket option. We propose a decentralized solution, and envision the operating system automatically assigning flows to the optimal interface, aided at most by simple hints from

applications.

Much recent work has argued that the multiple network connectivity options available to today's mobile devices is a blessing, not a curse. Johansson et al. [14], among others, show how Bluetooth radios are often preferable to IEEE 802.11 for short-range, low-power communication. Bahl et al. [1], illustrate scenarios where multiple radios can help devices save energy, enhance their data communication capacity, make wireless AP handoff more seamless, and better tolerate wireless link problems. Draves, et al. [9], show how overall throughput can be increased for multi-radio nodes in mesh networks, by dynamically choosing the "best" outbound link when forwarding a given packet. Stemm and Katz [32] were some of the first to recognize the hierarchical nature of overlapping wireless networks. Much like cache hierarchies in computer architecture, multiple wireless networks commonly cover one spot, with the utility (e.g., bandwidth) of a network usually inversely proportional to its coverage radius.

## 7. STATUS AND CONCLUSION

We are currently applying our ideas to two systems: Thunderbird, Mozilla's mail client, which supports the IMAP [21] protocol, and BlueFS [26], a distributed file system that supports consumer electronics devices in a mobile context. These have been educational—in both cases, the applications make more stringent assumptions about order and connectivity than are necessary. It is not yet clear to us whether these assumptions reduce complexity, or were made only because they were originally designed for an infrastructure that supports infrequent mobility at best.

Each of these are connection-oriented, and each expects only a single connection per client machine/server pair. However, each also provides traffic that could be delivered opportunistically. Thunderbird periodically fetches email headers/bodies in the background, and such transfers can be deferred. Both Thunderbird and BlueFS can defer updates—outbound email messages and updated or created files—to times of high bandwidth. Thunderbird currently assume that network messages are ordered, though weaker semantics appear possible. BlueFS allows some messages to be reordered. As we gain experience with our proposed set of interfaces, we expect to develop a set of design rules to help applications fit the model of diversity and change in networking capability.

It is clear to us that designing for a single point in the network space—wired, WiFi, Bluetooth, etc.—is no longer acceptable for truly mobile computing. Rather, one must accept that networks of widely varying capability will be available, and should be exploited whenever necessary. Doing so requires a collaboration on the part of system and application. System support provides the means to discover and characterize new networking options as they become available, and predict near- to medium-term connectivity when possible. Applications must provide semantic information to best match data flows with this diverse set of connectivity options, as well as provide the mechanisms to switch between them. Over time, we expect to develop several new models of application communication to best take advantage of these environments.

## 8. REFERENCES

- [1] Paramvir Bahl, Atul Adya, Jitendra Padhye, and Alec Walman. Reconsidering wireless systems with multiple radios. *ACM SIGCOMM Computer Communication Review*, 34(5):39–46, October 2004.
- [2] Bay area wireless users group. <http://bawug.org/>.
- [3] Mauro Brunato and Danilo Severina. WilmaGate: A new open access gateway for hotspot management. In *Proceedings of the Third ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots (WMASH)*, pages 56–64, Köln, Germany, September 2005.
- [4] Casey Carter, Robin Kravets, and Jean Tourrilhes. Contact networking: A localized mobility system. In *Proceedings of the First International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 145–158, San Francisco, California, USA, May 2003.
- [5] R. Chandra, P. Bahl, and P. Bahl. MultiNet: Connecting to multiple IEEE 802.11 networks using a single wireless card. In *Proceedings of INFOCOM*, pages 882–893, March 2004.
- [6] General Electric Co. Ge healthcare and sprint deliver enhanced connectivity to hospitals across north america. Press Release, August 2007.
- [7] S. Czerwinski, B. Zhao, T. Hodes, A. Joseph, and R. Katz. An architecture for a secure service discovery service. In *Proceedings of the Fifth International Conference on Mobile Computing and Networking (MobiCom)*, pages 24–35, Seattle, Washington, USA, August 1999.
- [8] E. de Lara, Y. Chopra, R. Kumar, N. Vaghela, D. S. Wallach, and W. Zwaenepoel. Iterative adaptation for mobile clients using existing APIs. *IEEE Transactions on Parallel and Distributed Systems*, 16(10), October 2005.
- [9] Richard Draves, Jitendra Padhye, and Brian Zill. Routing in multi-radio, multi-hop wireless mesh networks. In *Proceedings of the Tenth International Conference on Mobile Computing and Networking (MobiCom)*, pages 114–128, Philadelphia, Pennsylvania, USA, September 2004.
- [10] Elias C. Efstathiou and George C. Polyzos. A peer-to-peer approach to wireless LAN roaming. In *Proceedings of the First ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots (WMASH)*, pages 10–18, San Diego, California, USA, September 2003.
- [11] K. Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 27–34, Karlsruhe, Germany, August 2003.
- [12] Adrian Friday, Nigel Davies, Nat Wallbank, Elaine Catterall, and Stephen Pink. Supporting service discovery, querying and interaction in ubiquitous computing environments. *Wireless Networks*, 10(6):631–641, November 2004.
- [13] T. J. Giuli, D. Watson, and K. P. Venkatesh. The last inch at 70 miles per hour. *IEEE Pervasive Computing*,

- 5(4):20–27, October–December 2006.
- [14] P. Johansson, R. Kapoor, M. Kazantzidis, and M. Gerla. Personal area networks: Bluetooth or IEEE 802.11? *International Journal of Wireless Information Networks*, 9(2):89–103, April 2002.
- [15] S. Kandula, K. C.-J. Lin, T. Badirkhanli, and D. Katabi. FatVAP: Aggregating AP backhaul capacity to maximize throughput. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 89–103, San Francisco, CA, April 2008.
- [16] R. H. Katz and E. A. Brewer. The Case for Wireless Overlay Networks. In *SPIE Multimedia and Networking Conference*, January 1996.
- [17] S. Lu, S. Park, C. Hu, X. Ma, W. Jiang, Z. Li, R. A. Popa, and Y. Zhou. MUVI: Automatically inferring multi-variable access correlations and detecting related semantic and concurrency bugs. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles*, Stevenson, WA, October 2007.
- [18] N. Marmasse and C. Schmandt. A user-centered location model. *Personal and Ubiquitous Computing*, 6(5–6):318–321, December 2002.
- [19] Yasuhiko Matsunaga, Ana Sanz Merino, Takashi Suzuki, and Randy Katz. Secure authentication system for public WLAN roaming. In *Proceedings of the First ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots (WMASH)*, pages 113–121, San Diego, California, USA, 2003.
- [20] Allen K. Miu and Paramvir Victor Bahl. Dynamic host configuration for managing mobility between public and private networks. In *Proceedings of the Third USENIX Symposium on Internet Technologies and Systems (USITS)*, pages 147–158, San Francisco, California, USA, March 2001.
- [21] D. Mullet and K. Mullet. *Managing IMAP*. O’Reilly, 2000.
- [22] J. Newsome and D. Song. Dynamic taint analysis: Automatic detection, analysis, and signature generation of exploit attacks on commodity software. In *Proceedings of the 12th Network and Distributed Systems Security Symposium*, San Diego, CA, February 2005.
- [23] E. B. Nightingale, D. Peek, P. M. Chen, and J. Flinn. Parallelizing security checks on commodity hardware. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, Seattle, WA, March 2008.
- [24] E. B. Nightingale, K. Veeraraghavan, P. M. Chen, and J. Flinn. Rethink the sync. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, Seattle, WA, November 2006.
- [25] NYCWireless. <http://nycwireless.net/>.
- [26] D. Peek and J. Flinn. EnsemBlue: integrating distributed storage and consumer electronics. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 219–232, Seattle, WA, November 2006.
- [27] C.E. Perkins. Mobile networking through Mobile IP. *IEEE Internet Computing*, 2(1):58–69, January–February 1998.
- [28] Naouel B. Salem, Jean-Pierre Hubaux, and Markus Jakobsson. Reputation-based Wi-Fi Deployment Protocols and Security Analysis. In *Proceedings of the Second ACM International Workshop on Wireless Mobile Applications and Services on WLAN Hotspots (WMASH)*, pages 29–40, Philadelphia, Pennsylvania, USA, October 2004.
- [29] M. Satyanarayanan. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*, 8(4):10–17, August 2001.
- [30] SeattleWireless. <http://seattlewireless.net/>.
- [31] Alex Snoeren, Hari Balakrishnan, and Frans Kaashoek. Reconsidering Internet mobility. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS)*, pages 41–46, Elmau/Oberbayern, Germany, May 2001.
- [32] Mark Stemm and Randy H. Katz. Vertical handoffs in wireless overlay networks. *Mobile Networks and Applications*, 3(4):335–350, December 1998.
- [33] D. S. Wallach, D. Balfanz, D. Dean, and E. W. Felten. Extensible security architectures for Java. In *Proceedings of the 16th Symposium on Operating Systems Principles*, Saint-Malo, France, October 1997.
- [34] Xinhua Zhao, Claude Castelluccia, and Mary Baker. Flexible network support for mobility. In *Proceedings of the Fourth International Conference on Mobile Computing and Networking (MobiCom)*, pages 145–156, Dallas, Texas, USA, 1998.