On Redundant Multipath Operating System Support for Wireless Mesh Networks

Yair Amir[1], Claudiu Danilov[2], Michael A. Kaplan[1], Raluca Musăloiu-Elefteri[1], Nilo Rivera[1]

[1] Johns Hopkins University, Baltimore, MD.
{yairamir, kaplan, ralucam, nrivera}@dsn.jhu.edu

[2] Boeing Phantom Works, Seattle, WA.
claudiu.b.danilov@boeing.com

Technical Report CNDS-2007-02 - December 2007
http://www.dsn.jhu.edu

Abstract

Low-cost wireless routers are changing the way people connect to the Internet. They are also very cheap, albeit quite limited, Linux boxes. These attributes make them ideal candidates for wireless mesh routers. In wireless mesh networks, redundant multipath is essential for achieving fast, lossless handoff as mobile devices roam throughout the coverage area. Implementing it in user level on low-cost wireless routers greatly degrades the achievable throughput due to CPU limitations. This paper presents a novel mechanism for redundant multipath routing in kernel space. We show an architecture that integrates this mechanism in a wireless mesh system, resulting in a high-throughput 802.11 mesh network with fast handoff over low-cost routers.

1 Introduction

Low-cost wireless routers are revolutionizing the way people connect to the Internet. The ease of deployment at home or office on one hand, and the good Internet connectivity they provide on the other, have made these wireless routers ubiquitous. These routers are also a revolution from another, less known perspective: They are very cheap, albeit quite limited, Linux boxes (around $50 a piece). These attributes make them very attractive and convenient for developers to implement their own applications.

Wireless mesh networks extend the connectivity area of mobile devices beyond the limited range of a single access point. They are also very affordable when implemented with off-the-shelf low-cost wireless routers. These networks can be easily deployed inside a building, campus, or on a large geographical area without requiring every access point to be physically connected to the Internet. A lot of research over the last few years has focused on making wireless mesh networks a reality, including MIT Roofnet [10], Microsoft MCL [1], and UCSB MeshNet [15].

In wireless mesh networks, mobile devices (laptops, PDAs) can freely roam throughout the wireless coverage area, potentially changing the connected access point frequently. Redundant multipath routing (i.e., the ability to send the same packet over multiple routes) during handoffs is essential for achieving uninterrupted connectivity ([19], [13], [4]) as mobile devices move from one access point to another. Other applications can also benefit from redundant multipath routing (Section 5). However, redundant multipath is not natively supported by current operating systems, limiting the routing mechanisms that can be used in these networks to user-level implementations.

In this paper, we propose a new approach to support redundant multipath forwarding in kernel space. To support client mobility, multipath routes are managed in user-space with multicast trees that allow for in-kernel multi-source multi-destination routing of unicast packets. The routing service encodes the mesh-source of each data packet at the mesh entry point, and instructs the kernel to route packets according to this information. Our solution does not add any overhead to data packets, utilizing existing space on the IP packet header instead.

The contributions of this work are: **(1)** A novel mechanism for redundant multipath routing in kernel space. **(2)** An architecture that integrates kernel-level redundant multipath routing in a wireless mesh system, resulting in the first high-throughput 802.11 mesh with fast handoff over low-cost routers. **(3)** A publicly available open-source implementation of the kernel modules for Linux.

2 Related Work

The routing process involves computing routes to a destination, usually taking into account the distributed and dynamic nature of the underlying network, and the actual forwarding of the packets. Packet *routing*, which specifies the rules on how packets will be forwarded, is commonly performed in user space, allowing different protocols to be easily deployable and upgradable (OSPF [18], RIP [12]). Packet *forwarding*, on the other hand, typically resides in the kernel to forward packets as fast as possible. Thus, routing relies on the forwarding capabilities provided by the operating system. This approach allows operating systems to be both flexible and efficient.

Previous work has looked into bypassing the kernel from the communication path [20]. However, it requires a sophisticated architecture, in addition to special support from the hardware. A common approach to extend the routing services provided by the kernel is to build user-level routers that forward packets at application level. RON [6] uses this approach to route packets through an overlay network to increase the reliability of the end-to-end path. End-System-Multicast [14] and Spines [3] also route through an application router to support overlay multicast without infrastructure support.

Other work has looked into operating system support for wireless ad-hoc routing protocols. Chakeres and Belding showed in [9] an in-kernel design and implementation of the ad-hoc AODV protocol using Netfilter modules, and showed performance improvement compared to user-level ad-hoc protocols. Kawadia et al. [16] proposed a complete architecture to support ad-hoc protocols in-kernel and a generic ad-hoc support library for user-level programs to control different ad-hoc protocols.

SMesh [4] recently showed how overlay multicast can be used in wireless mesh networks to provide fast handoff to unmodified 802.11 clients that connect transparently using DHCP. SMesh allows *multiple* access points to service the client during handoff, as it works in ad-hoc mode. In SMesh, packets sent by the mobile client are diverted from the kernel to the Spines user-level overlay router. Multicast trees are calculated in a way similar to that of MOSPF [17]. SMesh encapsulates client packets and sends them through the overlay network to the access points serving the destination. Once the packets are received by the destination's access points, SMesh strips the overlay headers and forwards the original packet to the mobile client using a raw socket. In our experiments we used the SMesh wireless mesh system as a testbed to implement and evaluate the benefits and drawbacks of the approach presented in this paper.

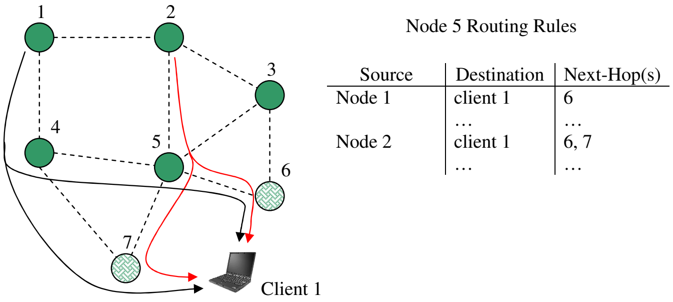
Redundant multipath is general in context and can assist a number of routing protocols. Handoff is only one of several uses of redundant multipath. Other applications will be discussed in Section 5.

3 OS Support for Redundant Multipath Routing

User-level overlay routing allows users to implement any protocol without requiring any special support from the kernel. The SMesh system uses this approach to implement redundant multipath, essential for achieving a seamless handoff. While very convenient, routing the entire traffic through user space becomes problematic for low-cost wireless routers which have limited processing power. It is widely known that forwarding packets through user space results in higher CPU utilization when compared to kernel space. The overhead can be attributed to two primary factors: memory copies and context switches. Each routing node must copy the packets from kernel space to user space in order to determine the next hop. After a routing decision is made, the packet must be returned to kernel space where it is sent on the network. That is, the user-kernel boundary must be crossed a minimum of two times per hop.

3.1 Architecture

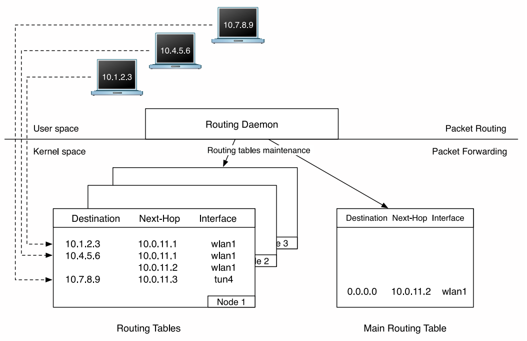
In wireless mesh networks, it is possible for packets to start flowing in the mesh from different sources. Several Internet gateways may coexist in multi-homed wireless mesh networks [5], any of which may need to forward packets to the client. Also, since clients may communicate with other clients in the mesh network, virtually every access point is capable of being the source of packets in the mesh. To provide optimal redundant multipath routing in these



Node 5 Routing Rules

Source	Destination	Next-Hop(s)
Node 1	client 1	6
	…	…
Node 2	client 1	6, 7
	…	…

Figure 1: The routes to a mobile client (multipath routing).



Destination	Next-Hop	Interface
10.1.2.3	10.0.11.1	wlan1
10.4.5.6	10.0.11.1	wlan1
	10.0.11.2	wlan1
10.7.8.9	10.0.11.3	tun4

Destination	Next-Hop	Interface
0.0.0.0	10.0.11.2	wlan1

Figure 2: Architecture.

networks, each node must consider the mesh source and the destination of each packet in order to determine the appropriate forwarding rule for that packet. In multicast, this is known as multi-source multi-destination multicast routing[1]. Figure 1 shows how packets are forwarded to the mobile client from two different sources (node 1 and node 2)[2]. Note that node 5 must forward the packets differently depending on the source of the packet.

Based on the description above, it is clear that a node must decide what are the next hops for a packet based on the mesh *entry point* as well as the destination address of the packet. However, the entry point cannot be determined by just looking at the packet destined to the client. One solution is to tunnel each packet from the entry point in the mesh to the mobile client. However, we need to instruct the kernel to remove the tunnel in the last hop, right before sending the packet to the client, which requires new kernel functionality. Otherwise, the mobile client may discard these packets. Another less obvious solution is to encode the mesh entry point in some of the existing space in the IP header of the packet. Specifically, we can encode the IP address of the entry point into the identification field from the IP header (also referred to as IPID). This is a 16-bit field used to identify the fragments of the IP datagrams. Together with the offset field, it is used by the IP layer to reassemble the fragmented datagrams. As the packet travels in the network, the intermediate routers leave the IPID field unchanged. To make a distinction between the original source (entry point in the mesh) and the rest of the nodes (routers) along the path, we use a bit from *type of service* (TOS) field, specifically, the *cost* bit. This approach benefits from no overhead in the mesh, in terms of packet size, and from the fact that no modification to the packet is necessary except at the entry point of the mesh.

Even if very convenient, modifying the IPID of the packets may create problems in the case of fragmented traffic. Current studies show that IP packet fragmentation is not commonly used today, and it amounts to between 1 and 2% [8] of the overall traffic. While advocating for or against the use of fragmentation is outside the scope of this paper [11], we choose to ignore the mesh entry point when the packet is fragmented, and forward it through a single path. Other solutions to encode both the fragmentation and the source node are possible. However, considering the small amount of the fragmented traffic, we believe this is a practical way for our system to support it.

Our approach to routing packets in the kernel is as follows: We first define a routing table in the kernel for each access point in the mesh, i.e., for each possible mesh entry point. In each table, we add an entry for each possible destination, i.e., for each client (Figure 2). This entry may include several next-hops, depending on the multicast trees determined by our routing daemon. Then, we instruct the kernel to encode in each packet the mesh entry point when the packet is first seen in the mesh network. Each node looks for this information at each incoming packet to select what routing table to use. Then, the kernel forwards the packet according to the entry that has the client address as the destination. For the example presented in Figure 1, router 5 will use different forwarding tables if the packets come from source 1 or source 2. In addition, to route packets from the clients to the Internet, each node sets in its main

[1] Using an IP-multicast tunnel is not a viable solution as it suffers from lower reliability due to the lack of 802.11 link-layer retransmissions. An alternative is to use IP-multicast with a *unicast tunnel* on each hop, but this approach incurs additional space in the packet as well as processing overhead on each node.

[2] Note that these mesh nodes are not the actual sources. Rather, they are the nodes that first received the packet in the mesh network, either from Internet or from clients connected to them.

routing table the route to the closest Internet gateway.

3.2 Implementation

There are several challenges in implementing such an architecture with the current services provided by the Linux kernel networking stack. Essentially, we need to be able to forward packets simultaneously on multiple paths to the same destination. Fortunately [3] , since version 2.2, the Linux kernel supports defining multiple routing tables and permits policy routing (a.k.a. rule based routing), which allows selecting different routing tables based on criteria other than the destination address. In our case, each mesh node maintains a routing table for each entry point in the network, and includes in each routing table an entry for each multicast group. That is, one routing table corresponds to all multicast trees that have that node as a source.

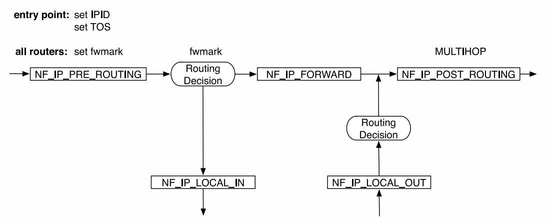


Figure 3: Implementation in Linux.

To alter the IPID field of the IP header, we wrote a simple Netfilter kernel module. The selection itself of which routing table to use, given the IP encoded in IPID, is done with policy routing using *fwmark*, a mark carried by the kernel as the packet travels through the kernel stack. Note that all Netfilter rules required to alter IPID field and to set *fwmark* and use it in table selection are added/deleted at run-time, as all possible entry points are not known in advance.

Normally, a routing table specifies a single forwarding action to be taken in a deterministic manner for a given packet. The `CONFIG_IP_ROUTE_MULTIPATH` option in the kernel configuration permits specifying several alternative paths for a destination. If no weight if given, the kernel considers all these paths to be of equal cost and chooses in a non-deterministic way which *one* to use when a packet arrives. Instead, we would like to send the packet to multiple nodes *simultaneously*, if the multicast tree indicates that. Therefore, we wrote a Netfilter target module, called MULTIHOP, which sends a copy of the packet to each next-hop found in the routing rule for a given destination. In order to use this module, one needs to recompile the kernel to export a function required to access the routing table (*fib_lookup*). Other than that, no changes are required in the kernel. The module is available for download from our website.

Figure 3 shows the path of a packet through the Linux kernel and the places where it interacts with our scheme. Immediately after the packet gets in, the entry point of the mesh must change the IPID field and set the TOS bit. Both the entry point and any intermediate router set the *fwmark* when processing a packet for routing [4]. We use the `NF_IP_PRE_ROUTING` Netfilter hook to do these modifications. The packet is then passed back to the kernel networking stack, where it goes into the routing mechanism in which the *fwmark* is used to choose the appropriate routing table. After the routing decision is taken, but before leaving the interface, the packet reaches the MULTIHOP module. Additional copies of the packet are created if there are more than one next-hop in the route. The module will simply exit if there is only one next-hop. We register the MULTIHOP module at the `NF_IP_POST_ROUTING` Netfilter hook, such that there will be no routing decisions afterward.

4 Experimental results

We evaluate our kernel redundant multipath scheme using low-cost Linksys WRT54G wireless routers running with a third-party OpenWRT firmware [2]. As this firmware was initially built using Linux kernel 2.4, we implemented our kernel modules for this version of Linux. Some of the experiments were performed using only wired connections, showing the CPU limitation in routers' performance, while other were performed in a wireless testbed of 17 nodes deployed among three buildings in our campus. In these experiments, the transmit power was set to $50mW$, and the long retransmission limit to 4.

[3] One of our goals is also to do as little changes as possible to the kernel, if any at all.

[4] *fwmark* is an internal mark in the kernel's packet data structure, and does not involve changing the packet as in the case of IPID and TOS.

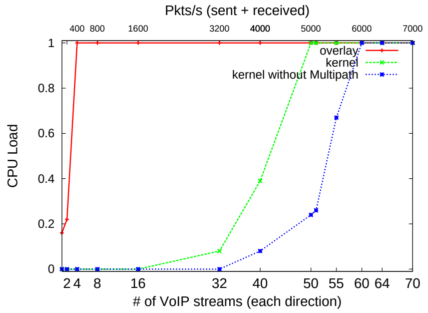


Figure 4: Average CPU load.

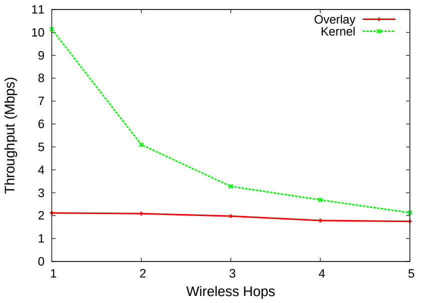


Figure 6: Average TCP throughput.

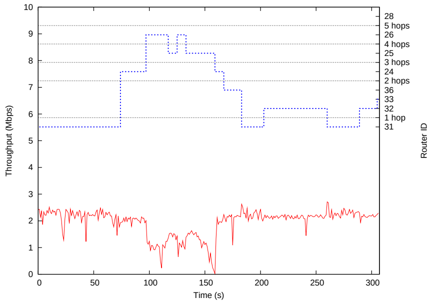


Figure 8: TCP throughput: Overlay (moving).

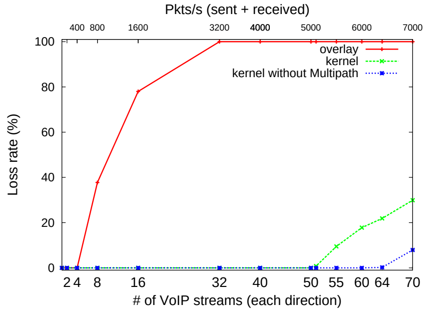


Figure 5: Loss rate.

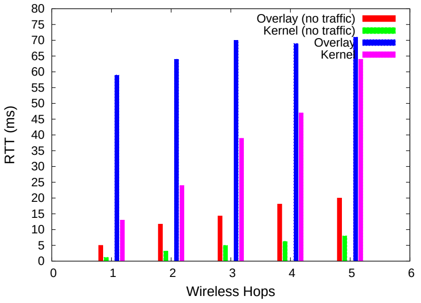


Figure 7: Average RTT.

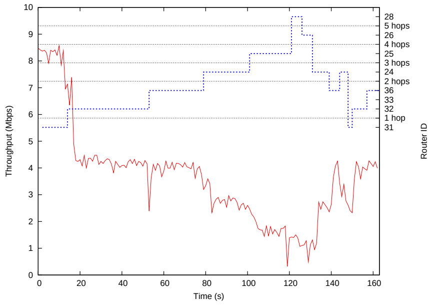


Figure 9: TCP throughput: Kernel (moving).

Overlay vs Kernel CPU comparison test: The first experiment demonstrates the CPU limitation and its effect on the packet loss rate as the transmission rate increases (Figures 4 and 5). To avoid losses caused by the wireless link-layer contention, we performed this test connecting the client to the router with a network cable. We sent 160-byte UDP packets, emulating VoIP streams, at rates corresponding to an increasing number of full-duplex VoIP streams between the client and the Internet. Each (one-way) stream has a rate of 64Kbps. We can see that with the overlay implementation, the CPU starts to be saturated at 400 pkts/s (4 full-duplex VOIP streams, or 512Kbps), in our kernel implementation at $5,000$ pkts/s (50 streams or aprox 6.4Mbps) while in kernel "without redundant multipath" implementation at $6,000$ pkts/s (60 streams or aprox 7.6Mbps). In each of these three scenarios, after a while, the loss rate starts to be non-zero: less than 8 streams (1Mbps) for overlay, 51 streams (11Mbps) for kernel and 64 streams (13Mbps) for kernel without our additions.

Overlay vs Kernel TCP throughput and RTT test: This experiment evaluates the *maximum* throughput that can be achieved in a multi-hop wireless network when forwarding through user and kernel space with our modifications. We connected 5 Linksys WRT54G routers in a simple "line" topology, and measured the TCP throughput while sending traffic from Internet to the client. Note that this continues to be a very controlled test. We only use one client, and we do not use background traffic to influence our results, as our goal is to obtain the throughput upper bound. We performed tests with the client placed 1, 2, 3, 4 and 5 hops away from the Internet gateway. The throughput results are presented in Figure 6. We also measured the round trip time (RTT) for both, overlay and kernel routing, with and without background traffic (Figure 7). TCP throughput for 1 hop improved from a CPU-limited amount of 2.1Mbps to a bandwidth limited amount of about 10Mbps in our setup. The latency of routing packets is also improved, as shown in the RTT experiment. In a separate experiment (not shown) with a full-duplex VoIP stream, packet latency for 1 hop improves from 10.5ms with overlay routing to 3.7ms with our kernel implementation for 80% of the packet.

Overlay vs Kernel in the deployed testbed: Figures 8 and 9 present the TCP throughput achieved over time in both overlay and kernel modes while moving throughout our 17 node wireless mesh network. Note that, as the two experiments were done sequentially to test the maximum achievable throughput, the sequence of handoffs could not be replicated between the two runs, even though the mobility pattern was similar. The mesh network opportunistically associates the mobile device to access points depending of current conditions. In order to see how far we are from the Internet gateway, we plot with a dotted line the access point that currently services the client. The horizontal lines mark when the number of hops increases by one. To simplify the graphs, we included only the routers that were involved in handling the client. With the overlay routing, the throughput is just above 2Mbps if the client is 1 or 2 hops away from the Internet gateway (routers 31 and 32, 33 and 36). This is consistent with the throughput reported in the previous test. As the number of hops increases, the throughput drops to 1Mbps and even lower. In the kernel routing, the throughput was about 8.5Mbps for 1 hop access points, 4.3Mbps for 2 hops and it drops to 1Mbps when the client is 6 hops away from the gateway (router 28).

5 Additional Applications

Traditionally, redundancy is equivalent to resiliency, allowing the entire system to function as a whole when some of its components fail. In the case of routing, existing protocols are self-healing such that, upon detection of a route failure, the protocol will search and/or establish a new routing path if one exists. The problem lies in the process of detecting the failure, which usually involves a timeout to expire after a relatively large amount of time. While reducing these timeouts is not desirable in routing protocols, as they may generate dangerous oscillations, we believe that using redundant multipath forwarding in case of uncertainty offers a viable solution for resilient routing.

With our redundant multipath scheme, wireless networks could route each packet with a different level of resilience. Consider the network shown in Figure 10 where the source can reach the destination through one or more paths. These paths may be disjoint, and the number of paths may depend on the current state of the network and the levels of resilience needed (note that these routes cannot be managed by multicast trees). The graph shows the paths for three possible levels of redundancy. Utilizing our approach, different routing tables can have entries that correspond to different levels of redundancy. Then, each packet can carry its desired reliability level in its IPID so that each router is able to forward the packet using the appropriate forwarding table.

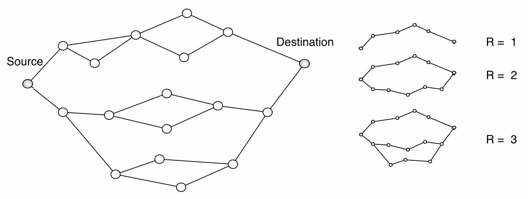


Figure 10: Generic Redundant Multipath.

Redundant multipath can also be used to send time-sensitive information in hostile environments where nodes may be compromised by an adversary. While existing byzantine routing protocols [7] try to detect bad links and avoid routing through compromised nodes, this detection will take some time. With redundant multipath, the redundancy level of packets can be based on the importance of the data as well as on the current threat level in the working environment.

6 Conclusion

This paper introduced a mechanism that achieves redundant multipath routing in kernel space. The architecture was integrated in a real system consisting of low-cost routers, resulting in a high throughput wireless mesh network with fast-handoff. We also showed that redundant multipath is a useful component to have in Unix-like kernels for other applications. Experimental results show that using the kernel-level redundant multipath service achieves a significant performance improvements both in throughput and latency. An open source implementation of the kernel modules described in this paper is available at www.smesh.org.

References

[1] Microsoft research networking research group. http://research.microsoft.com/mesh.

[2] OpenWrt. http://openwrt.org.

[3] Yair Amir and Claudiu Danilov. Reliable communication in overlay networks. In *Proceedings of the IEEE DSN*, pages 511–520, June 2003.

[4] Yair Amir, Claudiu Danilov, Michael Hilsdale, Raluca Musaloiu-Elefteri, and Nilo Rivera. Fast handoff for seamless wireless mesh networks. In *MobiSys 2006*, pages 83–95, New York, NY, USA, 2006. ACM Press.

[5] Yair Amir, Claudiu Danilov, Raluca Musaloiu-Elefteri, and Nilo Rivera. An inter-domain routing protocol for multi-homed wireless mesh networks. *International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2007), Helsinki, Finland*, June 2007.

[6] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proc. of the 18th Symposium on Operating Systems Principles*, pages 131–145, October 2001.

[7] Baruch Awerbuch, David Holmer, Cristina Nita-Rotaru, and Herbert Rubens. An on-demand secure routing protocol resilient to byzantine failures. In *WiSE '02: Proceedings of the 1st ACM workshop on Wireless security*, pages 21–30, New York, NY, USA, 2002. ACM Press.

[8] K.C. Claffy C. Shannon, D. Moore. Beyond folklore: Observations on fragmented traffic. *IEEE/ACM Transactions on Networking*, 2002.

[9] Ian D. Chakeres and Elizabeth M. Belding-Royer. Aodv routing protocol implementation design. In *ICDCSW '04: Proceedings of the 24th International Conference on Distributed Computing Systems Workshops - W7: EC (ICDCSW'04)*, pages 698–703, Washington, DC, USA, 2004. IEEE Computer Society.

[10] Benjamin A. Chambers. The grid roofnet: a rooftop ad hoc wireless network. Master's thesis, Massachusetts Institue of Technology, May 2002.

[11] Girish P. Chandranmenon and George Varghese. Reconsidering fragmentation and reassembly. In *PODC '98: Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*, pages 21–29. ACM Press, 1998.

[12] C. Hedrick. Routing Information Protocol. *RFC1058*, June 1988.

[13] Ahmed A.-G. Helmy, Muhammad Jaseemuddin, and Ganesha Bhaskara. Multicast-based mobility: A novel architecture for efficient micromobility. *IEEE Journal on Selected Areas in Communications*, 2004.

[14] Yang hua Chu, Sanjay G. Rao, and Hui Zhang. A Case For End System Multicast. In *Proceedings of ACM SIGMETRICS*, June 2000.

[15] G. Chandranmenon S. Miller K. Almeroth E. Belding-Royer K. Ramachandran, M. M. Buddhikot. On the design and implementation of infrastructure mesh networks. *WiMesh*, 2005.

[16] Vikas Kawadia, Yongguang Zhang, and Binita Gupta. System services for ad-hoc routing: Architecture, implementation and experiences. In *MobiSys 2003*, pages 99–112, New York, NY, USA, 2003. ACM Press.

[17] J. Moy. Multicast extensions to OSPF. RFC 1584, IETF, March 1994.

[18] J. Moy. OSPF Ver 2. *RFC2328*, April 1998.

[19] S. Seshan, H. Balakrishnan, and R. Katz. Handoffs in Cellular Wireless Networks: The Daedalus Implementation and Experience. *Kluwer Journal on Wireless Personal Communications, 1996.*, 1996.

[20] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-net: a user-level network interface for parallel and distributed computing. In *SOSP*, pages 40–53, New York, NY, USA, 1995. ACM Press.