Low-Overhead Routing for High-Performance Wireless Mesh Networks

Independent Study, 600.810.12, Dr. Yair Amir

Michael Kaplan
kaplan@dsn.jhu.edu

June 16, 2006

Abstract

Routing, a task ordinarily performed solely by the operating system, has since crossed over to the user-space where sophisticated overlay networks can make more informed next-hop forwarding decisions. While application routing often improves network performance, it does incur the cost of significant CPU processing when copying packets between kernel and application memory spaces. To date, overlays have primarily been intended for and deployed over end-host machines that are comparable to or better than regular PCs; thus additional computational overhead can be handled without hindering network performance. However, in many practical-purpose computing environments, Internet connectivity is deployed quickly, easily, and inexpensively using low-end commercial off-the-shelf (COTS) products. A 30-node mesh network of Linksys wireless routers, for example, can be used to provide connectivity over a large geographic area at a cost of under $2000, revolutionizing the way many typical users plan to build networks. The overlay is an essential component in the communication infrastructure for these networks, but unlike high-end devices, limited COTS products cannot transparently absorb the additional overhead. Throughput degrades dramatically when the routers do not simply forward through the kernel as is their intended manufactured purpose. In order to retain the benefits of an overlay without sacrificing performance in these weaker hardware environments, a new hybrid routing design is necessary that incorporates the best aspects of overlay and kernel approaches. We present the architecture of such a system based on the SMesh network and evaluate its benefit.

1 Introduction

Routing in overlay networks traditionally follows one of two well-established design paradigms: dynamic application-based protocols or static forwarding over predefined kernel tables. The application approach to routing tends to be more desirable since it enables the overlay to perform a number of smart algorithms that significantly improve connectivity and performance. Spines for instance, is a modern state-of-the-art overlay that provides link assessment, hop-by-hop reliability, and application multicast among a number of other flexible user-space protocols [1]. However, this technique does come with a price as it is highly CPU intensive to manipulate packets from outside the kernel. Because kernel and application memory spaces are logically separate, memory copies are required to move packets back and forth. This cost, nevertheless, is generally overlooked since the intended deployment is over high-end host machines that can easily handle the additional processing. Until recently, this hardware assumption was reasonable, but a developing trend in networking architecture now challenges this notion.

Often it is desirable to quickly and cost-effectively deploy a network that provides peer-to-peer and Internet connectivity for clients. In such networks, low-cost COTS equipment is especially useful as it can provide coverage over a large area at little expense. Figure 1 for example, depicts our mesh network of Linksys wireless routers spread across the Johns Hopkins computer science building, retailing at only $60 per router. Thus for a mere $2000, we can quickly build a fairly large 30-node system, revolutionizing the way network deployment is typically carried out. The mesh connectivity in our network is provided by two applications running on each router: SMesh and Spines. SMesh implements various management protocols to handle peer-to-peer connectivity, Internet connectivity, and fast-handoff for mobile clients across the mesh [2]. Spines serves as the generic overlay architecture which monitors the quality of virtual links and routes packets to the appropriate next-hop. In a

SMesh network, few of the nodes actually act as Internet Gateways, so it is expected that mobile clients require several wireless hops to obtain Internet access.

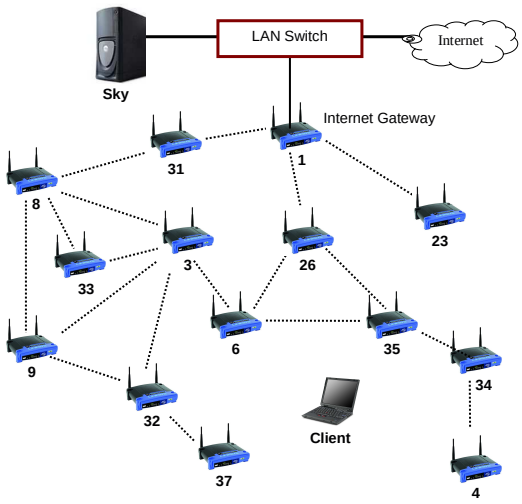


Figure 1: Sample SMesh configuration [2].

The low cost of the Linksys routers is made possible by their use of limited hardware - the routers feature only 200 MHz processors and 16 MB of RAM. While this hardware is sufficient for native packet forwarding, the devices are not well-equipped for computationally expensive user-space operations. SMesh runs well on better equipped machines, but because it relies on application routing, it faces a performance bottleneck when ported to the low-end routers. On a single wireless hop to the Internet gateway, the router is at nearly 100% CPU usage during data transfer: 58% SMesh and 38% Spines. With two wireless hops to the gateway, CPU usage is similarly dominated by the applications: SMesh 25% and Spines 35%. Nothing has actually improved for SMesh in the two-hop situation, but rather due to contention and exponential back off, the client is inhibited from transmitting at the same rate it did in the one-hop configuration.

Under these conditions, the transfer rate for mobile clients peaks at only 1-1.5 Mbps. Even between direct neighbors, in which the ideal transfer rate should be on the order of 15-18 Mbps, the rate is still unusually reduced due to the memory copies. To avoid this excess processing, the alternative is to switch back to a kernel-based approach in which the node routing tables are preconfigured to reflect the topology as is done in X-Bone[3], but unfortunately this technique reduces the degree of control needed to maintain a realistic network in which benign failures occur.

Therefore, we present a new routing paradigm, a hybrid approach that captures the best of both worlds. Our mechanism falls between the two extremes, using flexible application forwarding for control messages and low-overhead kernel routing for data packets. Under this scheme, the control information is used to continuously update the kernel tables in order to reflect the current overlay topology, allowing data packets to pass transparently without user-space interaction. Control messages still require essential application-level processing in order to maintain the mesh and since these transfers are less frequent and relatively small, the rate is acceptable as we do not incur high computational cost. We implemented this technique as an additional module in Spines and it achieves SMesh data transfer rates of nearly 6 Mbps for UDP over three wireless hops. The system has since become an integral part of SMesh as of version 1.2d.

2 Related work

Our research contributes to the field of overlay networks and wireless mesh networks. The primary benefit of an overlay is the ability to implement new services and make complex routing decisions on a subset of nodes to yield better performance than that of the underlying Internet. There are essentially two schools of thought with regard to overlay routing as it has been implemented in previous systems: application-based and kernel-based. Spines is one such system that performs significant application-level computations to improve connectivity for its nodes [1]. It actively measures link qualities and chooses paths based on distance, latency, or loss. Spines uses its own headers to provide overlay multicast, anycast, and hop-by-hop reliability. Additionally, Spines automatically reconfigures its daemons upon detecting benign failures.

Conversely, the X-Bone [3] takes a kernel-based routing approach. X-Bone provides a graphical tool which allows the user to specify an overlay topology from a set of participating nodes. The tool then interacts with a management daemon to disseminate the configuration. The affected nodes configure tunnels and set their kernel routing table to reflect the desired arrangement. No further application interaction is necessary, allowing packets to be forwarded using the native protocols. X-Bone, however, does not monitor or dynamically update the state of its links; thus a given configuration is static until further interaction with the GUI. While this approach may be less CPU intensive, we lose the degree of routing control needed to handle un-

predictable links, particularly in the wireless environment. Furthermore, X-Bone and similar kernel-based approaches are not capable of providing multicast.

In our system, we modify Spines to share the principle qualities of both options. Spines is inherently an application overlay, but our modification allows it to continuously update the kernel routing table to reflect the best path for data messages. Spines control messages continue to use the application-based protocols, but since they are small as compared to data, they do not incur significant computational cost.

Interestingly, RON [4] already provides a form of hybrid routing. In RON, packets are forwarded natively through the kernel-space until a disruption in service is detected. At this point the packets are intercepted by the overlay and application protocols are used to determine a better next-hop. Our solution, improves upon this idea since we always route through the kernel and do not depend on FreeBSD.

The MIT Roofnet [5] is one of the first mesh networks that deployed Internet connectivity over wireless nodes. The nodes consist of preconfigured PCs connected to an antenna mounted at the node's location. SMesh on the other hand, presents a more flexible approach. It is deployed over a series of Linksys wireless routers and allows any 802.11 device to connect to the network without any additional configuration by the client. Additionally, SMesh is the first to provide fast-handoff between access points as mobile clients traverse across the mesh. Roofnet yields throughput of 627 kbps while the original version of SMesh yields a slightly improved 1-1.5 Mbps. Nevertheless, both suffer from low transmission rates.

3 Overlay Infrastructure

The SMesh software facilitates the activity of clients as they access the network. SMesh handles the DHCP server that assigns IP addresses to its clients. The software is then used to direct client traffic and handle the selection of the clients best access point. This communication is based on our Spines messaging system which runs as a background daemon in conjunction with SMesh. Spines provides transparent multi-hop unicast, multicast and anycast communication between the wireless mesh nodes. SMesh instantiates two Spines multicast groups for each client: a data group and a control group. The groups are used to send their respective message types to multiple access points when a client is undergoing a *handoff*, switching

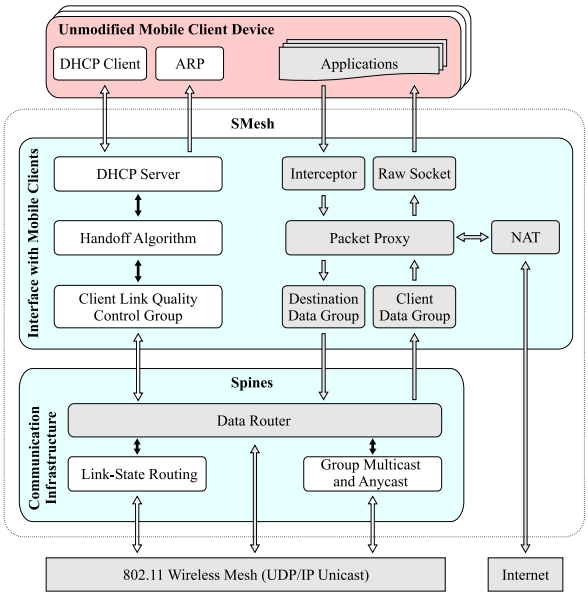


Figure 2: The original SMesh architecture [2].

between access points. SMesh also uses a Spines anycast group to direct traffic towards the Internet gateways. The routing in Spines is coordinated by two modules: Link-State routing, and Group Multicast/Anycast routing. The Link-State module runs the Floyd-Warshall algorithm on each received state update in order to compute the best next-hop between nodes. The multicast module maintains the node's neighbors to the groups and builds multicast trees as needed. As an optimization, Spines only builds multicast trees for groups to which messages are in the process of being sent.

4 Low-Overhead Routing

In an ideal situation, there is no need for SMesh or Spines to intercept the data packets at any point during the transfer. We can achieve near optimal transfer rates if data packets traverse only the kernel-space. In the original version of SMesh, such routing does not occur since the packets need to interact with Spines to reach the appropriate next-hop. However, since the routing knowledge of these applications is known beforehand from earlier control messages, we can use this information to run system calls that dynamically update the kernel routing tables to reflect that of the overlay. This routing scheme in effect offloads the routing responsibilities from the user-space to the kernel-space, providing low-overhead routing without sacrificing the benefits of an overlay. As desired, this new mechanism requires no memory copies at any node

during data transfer. Implementation requires a new module in Spines which sets the routing table upon receipt of relevant topology state packets. Section 4.1 describes the design of the kernel routing module and section 4.2 compares our changes to their counterparts in the original version.

4.1 Route Management

Internal Routes
Internal routes refer to the virtual links between Spines nodes participating in the network. These routes are used to forward packets between routers in the SMesh network. The appropriate paths can be easily detected at the completion of the Floyd-Warshall algorithm, which runs periodically in Spines to compute the routes to all Spines nodes. For all non-neighbors, a route must be set with the appropriate next-hop as the routing entry gateway. Neighbors do not require routes since they are within radio range of each other and will match the default subnet route, which in our network is 10.0.11.0/24 on interface eth1.

Client Routes (Access Points)
Client routes refer to the path required to reach the mobile clients in the mesh. In SMesh, clients are identified via unique multicast groups. When Spines on a given router invokes a join for a group of the form 227.x.x.x, this action indicates that the router has become the access point for that client. Spines must then set a route in which the destination and gateway are set to the clients IP address 10.x.x.x. This route allows the access point to deliver messages destined to its client. Conversely, if Spines invokes a leave for the multicast group, it is no longer the access point for the respective client and must remove the previously added route.

Client Routes (Intermediate Nodes)
Client routes must also be set on the intermediate nodes to reach the access point that is directly connected to the client. However, these routes are not trivial to compute. While Spines featured a periodic mechanism that facilitated updating internal routes, the client routes are dependent upon multicast trees which are not recomputed with state updates. Tree computation is expensive and therefore is only performed when a message is sent to the respective multicast group. However, in the kernel routing scheme, no *data messages* are ever sent to the multicast groups; they are only used in this case to identify the access points. To circumvent this Spines design feature, the multicast module was modified so that when it receives a group state packet, it processes the update and refreshes its direct neighbors to that multicast group. The neighbor with the lowest routing cost to the group is then used as the next-hop to the client, and the route is set in the kernel. This action is still less expensive than computing a complete multicast tree, but it should be noted that it does change the underlying Spines code.

Default Routes
Default routes refer to the last entry in the routing table which directs packets to a default next-hop when their destination does not match that of another entry. In SMesh, the default route must lead back to the Internet gateway so that client Internet activity can reach the external network. Internet gateways join a predefined anycast group. Using the technique described for client routes on intermediate nodes, upon processing a group state message for this specific anycast group, the closest anycast neighbor is retrieved and then set to be the node's default gateway. If the node later joins the anycast group, it will clear this default route. The router then automatically detects the network connected to the vlan1 interface and sets a default route to the server.

The routing entries are stored in a linked list and updated in response to topology changes. In all cases, new routes are added before old routes are deleted to reduce message loss. The implementation is also able to respond to crash failures since these events are topology changes which thus trigger a review of next-hop selections for affected routes. See figure 7 at the end of this report for a graphical representation of a sample network.

4.2 Functionality Mapping

Unicast/Multicast/Anycast
In the original version of SMesh, Spines is used as a message bus for data packets. Packets destined for a client are sent on its respective multicast data group. Packets destined for the Internet are sent on the SMesh anycast group. The kernel version, however, is a strictly unicast system for data transmission. We use the multicast and anycast groups in this case only to identify the access points. Messages sent to a client are sent directly, using the IP of the client as the packets destination. The membership of the anycast group is used to determine the closest Internet gateway, allowing nodes to set the appropriate default route. Packets destined for the Internet will traverse the default routes until reaching the Internet gateway.

Handoff
Spines multicast allows the original SMesh to send packets to the mobile client at multiple access points, effectively eliminating loss while the user undergoes handoff. As we do not provide multicast in the low-overhead version, there is an increased delay during handoff. First the new access point joins the data group, causing it to update its client route for direct forwarding. The join propagates through the network, potentially causing routing updates on the intermediate nodes. Eventually, one of the access points leaves the data group, replacing its direct route to the client with a best next-hop to the clients data group. The leave propagates through the network, again updating routes at intermediate nodes if necessary. The handoff is complete when the nodes reach a consistent view of the group-state information such that a routing path exists between the client and the other nodes. This delay can be observed during a series of pings to the outside network. We measured an approximate 5 second delay during most handoffs which is acceptable for many applications as it is typically less than the TCP retransmission timeout computed by the algorithm in [6] which empirically has been on the order of 30 seconds in our network.

Hop-By-Hop Reliability
Both the original and current systems are semi-reliable between hops. They rely only on 802.11 retransmissions to deliver packets even though Spines is capable of providing hop-by-hop reliability.

Multiple Gateways
Similar to that of the original system, kernel routing supports multiple gateways in the same connected component. Rather than simply send to the anycast group, we retrieve the closest neighbor to the anycast group and set it as the default gateway. Joins and leaves on the anycast group trigger updates on the default routes so that we are always using the closest Internet gateway.

Multi-homed Network
The kernel version of SMesh does not yet handle a multi-homed network; however, we do have a proposed solution. The multi-homed system consists of several SMesh networks connected over the Internet from their respective Internet gateways. When a mobile client moves between connected components, we wish to transfer the client's connections so that the inter-domain handoff is transparent. Our proposed solution requires that at configuration time, a single Internet gateway is selected as the primary gateway among the connected components. All other gateways will configure a GRE tunnel to the primary gateway and set the tunnel as its default route. All Internet connections will thus be rooted at the primary gateway, allowing a client to move between connected components without a need for inter-domain handoff. These changes require modifications to the SMesh executable script. Figure 3 depicts the hardware configuration for such a network.

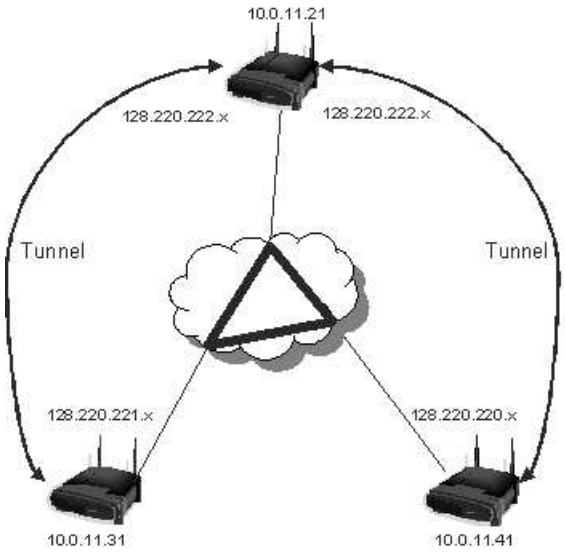


Figure 3: Multi-homed network with 10.0.11.21 set as primary gateway with GRE tunnels for kernel routing.

5 Performance Evaluation

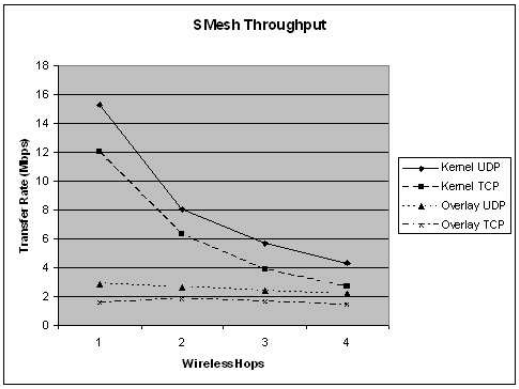


Figure 4: Throughput measurements for kernel and overlay SMesh. UDP sending rate for kernel and overlay versions were 20 Mbps and 3.125 Mbps respectively so as not to overload the receiver.

To benchmark the kernel routing performance, we deployed SMesh over four different networks consisting of one, two, three, and four Linksys routers respectively. We then measured the rate characteristics using iperf between a wireless 2GHz IBM Thinkpad and a wired Pentium III 500 MHz server connected via wire to the Internet gateway. Throughput measurements can be seen in figure 4. Each data point represents an average of five measurements taken in the indicated configuration. For TCP, we can see that kernel routing is 7.6 times faster on 1 wireless hop, 3.5 times faster on two wireless hops, 2.4 times faster on three wireless hops, and 1.9 times faster on four wireless hops. Similarly UDP over kernel routing yields rates faster than that of overlay links: 5.3 times on one hop, 3.1 times on two hops, 2.4 times on three hops, and double on four hops. We expect that typical SMesh clients will require two to three wireless hops to reach the nearest Internet gateway, thus TCP rates of 3.8 Mbps or 6.3 Mbps and UDP rates of 5.7 Mbps or 8.1 Mbps can be expected. We also notice that throughput decreases as the number of wireless hops increases when the routers are within radio range of each other. This degradation results primarily from contention among adjacent routers. While it may appear that overlay rates are unaffected by the different experimental configurations, rather it is the CPU bottleneck acting as a more dominating bound than is contention.

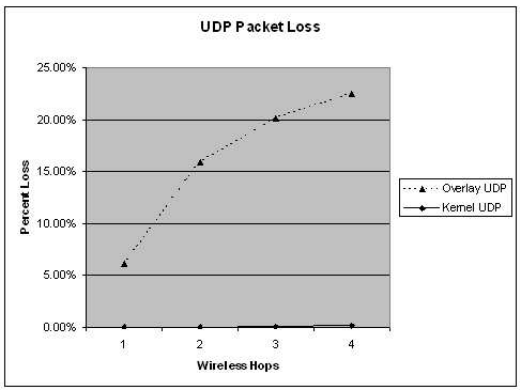


Figure 5: Loss rates with UDP sending rates of 20 Mbps in kernel mode and 3.125 Mbps in overlay mode.

We now consider the loss rates during the UDP transfers. Shown in figure 5, packet loss grows rapidly in the overlay routing scheme. Loss is apparently an indirect side-effect of the excess packet movement between application-space and kernel-space. Negligible loss is detected in kernel mode, occurring at only 0.2% on four wireless hops.

6 Current Limitation

The one drawback to our current approach is that we can no longer support application multicast which is needed to provide fast-handoff. Effort was spent attempting to manipulate the multicast routing table in a similar fashion as we do the unicast kernel table. However, since multicast packets are sent unreliably without 802.11 retransmission, the wireless environment would experience unacceptable loss rates during such transfers. Currently, we have a proposed solution which involves modifying the kernel to provide a type of multicast. Ordinarily the kernel forwards a message to the first match traversed in the routing table. Instead, we would want to enter multiple routes for the same destination and have the kernel forward the message over each matching entry. The messages are still sent unicast, but we could then send a given message to multiple recipients. For example, during a handoff between 10.0.11.30 and 10.0.11.40, 10.0.11.20 would be able to reach the client by sending unicast data to both 10.0.11.30 and 10.0.11.40. Further investigation is needed to determine if this is a viable approach. Figure 6 illustrates the routing table during this configuration.

Destination	Gateway	Genmask
10.189.214.61	10.0.11.40	255.255.255.255
10.189.214.61	10.0.11.30	255.255.255.255
10.0.10.0	*	255.255.255.0
10.0.0.0	*	255.0.0.0
default	10.0.11.10	0.0.0.0

Figure 6: Routing table for 10.0.11.20 providing multicast with proposed kernel modification.

7 Conclusion

In this study, we have presented an analysis and implementation of a low-overhead kernel routing mechanism for wireless mesh networks. Overlay routing on a wireless network is simply too costly on limited-hardware devices as the required memory management creates a CPU bottleneck that inhibits performance. We can obtain better performance when routing remains in the kernel. Yet we do not want to sacrifice the routing benefits of overlay networks. Thus, by dynamically updating the kernel routing table to reflect the overlay knowledge, we can achieve an effective balance that yields higher performance.

References

[1] Yair Amir and Claudiu Danilov. Reliable communication in overlay networks. In *DSN* [1], pages 511–520.

[2] Yair Amir, Claudiu Danilov, Michael Hilsdale, Raluca Musaloiu-Elefteri, and Nilo Rivera. Fast handoff for seamless wireless mesh networks. MobiSys, 2006.

[3] Joseph D. Touch. Dynamic internet overlay deployment and management using the X-bone. In *ICNP*, pages 59–68, 2000.

[4] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *SOSP*, pages 131–145, 2001.

[5] John C. Bicket, Daniel Aguayo, Sanjit Biswas, and Robert Morris. Architecture and evaluation of an unplanned 802.11b mesh network. In Tom La Porta, Christoph Lindemann, Elizabeth M. Belding-Royer, and Songwu Lu, editors, *MOBICOM*, pages 31–42. ACM, 2005.

[6] Rfc 2988. http://www.rfc-archive.org.

Appendix A: Software Documentation

The following files were added or modified in the Spines distribution:

kernel_routing.c and kernel_routing.h
The functions implemented in these files control the management of the kernel routing table. The same Route_Entry structure is used for client routes and internal routes, but the alive field is needed only for internal routes. When the Floyd-Warshall algorithm runs, internal routes are marked alive so long as they are still in use. Internal entries no longer alive will be deleted when Set_Internal_Routes is run. Function names indicate whether they are intended for client entries or internal entries. Functions that can operate on either entry type require a mode parameter indicating the type so that discrepancies between the two are handled.

multicast.c and multicast.h
In this module we modify the code to include a new function, Get_Closest_Mcast_Neighbor. The function first discards and repopulates the neighbor structure for the requested multicast or anycast group. This step is necessary to insure we are considering the most up-to-date group state information. It then determines the neighbor with the lowest routing cost to the group and returns this node as the closest multicast neighbor. This function is called at the end of Groups_Process_state_cell in order to update client routes when the multicast topology changes. Additionally, the Join_Group and Leave_Group functions are modified to update access point client routes.

route.c
In this module we modify the Set_Routes function to add routing entries during the progress of the Floyd-Warshall algorithm. If the route has not changed since the last run, the route is marked alive by the kernel routing module so that it remains in the table.

The following file was modified in the SMesh distribution:

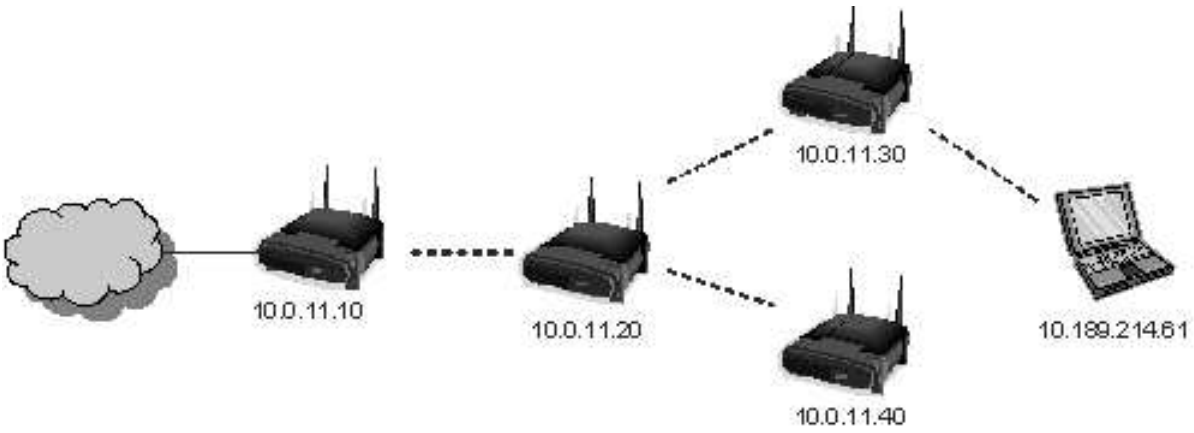
smesh_proxy.c
In this file, we add an if-clause to the packet interceptor. If the Kernel_Routing parameter is enabled, the libpcap socket is no longer used to intercept packets. If the parameter is disabled, SMesh continues to intercept packets as it did in the original version.

Appendix B: Installation Instructions

Compilation
To compile the low-overhead routing, simply use the build script present in the SMesh source directory. Set the BUILDSPINES and BUILDSMESH parameters to 1 and run the script.

Installation
After running the build script, the SMesh and Spines binaries appear in the source install directory. Copy the contents of the entire directory to /jffs/ on the Linksys routers.

Execution
Open the smesh.conf configuration file. Low-overhead routing can be disabled or enabled automatically in the configuration file by setting the KERNEL parameter to 0 or 1 respectively. When KERNEL is disabled, the routing will take place as it did originally in the user-space. To enable low-overhead routing manually, use the following flags on the executable: In SMesh, the -k option enables low-overhead routing. In Spines, -k1 enables kernel routing for internal routes and -k2 enables kernel routing for client routes. The -k1 option should always be used to allow transparent ssh between non-neighbors. All flags must be set to use low-overhead routing.



Routing Table for 10.0.11.10

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
10.0.11.30	10.0.11.20	255.255.255.255	UGH	0	0	0	eth1
10.0.11.40	10.0.11.20	255.255.255.255	UGH	0	0	0	eth1
10.189.214.61	10.0.11.20	255.255.255.255	UGH	0	0	0	eth1
128.220.221.0	*	255.255.255.0	U	0	0	0	vlan1
10.0.10.0	*	255.255.255.0	U	0	0	0	vlan0
10.0.0.0	*	255.0.0.0	U	0	0	0	eth1
default	commedia.cnds.j	0.0.0.0	UG	0	0	0	vlan1

Routing Table for 10.0.11.20

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
10.189.214.61	10.0.11.30	255.255.255.255	UGH	0	0	0	eth1
10.0.10.0	*	255.255.255.0	U	0	0	0	vlan0
10.0.0.0	*	255.0.0.0	U	0	0	0	eth1
default	10.0.11.10	0.0.0.0	UG	0	0	0	eth1

Routing Table for 10.0.11.30

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
10.0.11.10	10.0.11.20	255.255.255.255	UGH	0	0	0	eth1
10.0.11.40	10.0.11.20	255.255.255.255	UGH	0	0	0	eth1
10.189.214.61	10.189.214.61	255.255.255.255	UGH	0	0	0	eth1
10.0.10.0	*	255.255.255.0	U	0	0	0	vlan0
10.0.0.0	*	255.0.0.0	U	0	0	0	eth1
default	10.0.11.20	0.0.0.0	UG	0	0	0	eth1

Routing Table for 10.0.11.40

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
10.0.11.30	10.0.11.20	255.255.255.255	UGH	0	0	0	eth1
10.0.11.10	10.0.11.20	255.255.255.255	UGH	0	0	0	eth1
10.189.214.61	10.0.11.20	255.255.255.255	UGH	0	0	0	eth1
10.0.10.0	*	255.255.255.0	U	0	0	0	vlan0
10.0.0.0	*	255.0.0.0	U	0	0	0	eth1
default	10.0.11.20	0.0.0.0	UG	0	0	0	eth1

Figure 7: Routing tables for network depicted with client currently connected at 10.0.11.30.