

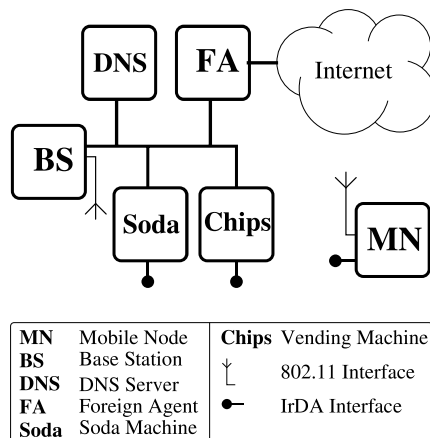


Fig. 5. Example network from Section 5

5. Examples

Some simple examples serve to illustrate the operation of Contact Networking. For these examples, we discuss a hypothetical user, Prashant, with a mobile node that has infrared, wireless LAN and wired Ethernet interfaces. Prashant is a Computer Science graduate student, who happens to be wandering around the CS building. The GRIP of Prashant's mobile node is topologically located on the Ethernet LAN. There is a building-wide wireless LAN, which has an attached DNS server and a MobileIP Foreign Agent, as in Figure 5. All machines on the CS building network are Contact Networking enabled.

1) *Simple Local Communication:* In the first scenario, Prashant is walking through the Computer Science building, when he passes a vending machine. Feeling a grumbling in his belly, Prashant decides to purchase a bag of Fritos™. We hypothesize the existence of software in Prashant's mobile node and the vending machine that is capable of automatically performing a small cash transaction. To initiate the transaction, Prashant points his device at the vending machine's infrared port and clicks the "pay" button. A flurry of events ensues.

The application on Prashant's machine tries to resolve the CNS alias `any.irda`. The IrDA-specific CNS module performs native IrDA discovery to detect the vending machine nearby. As a part of the discovery process, Prashant's node and the vending machine exchange CNS records containing their GRIPs and DNS names. As a side effect of CNS discovery, Contact Networking routes are created to enable on-demand connection of the two machines. The CNS resolver on Prashant's machine returns a `struct hostent` to the application, informing it that `any.irda` is in fact an alias for `chips.cs.uiuc.edu` with GRIP `128.174.244.91`.

Having obtained an IP address for the desired neighbor, Prashant's application initiates a TCP flow to the ap-

plication in the vending machine. The first packet of this flow is queued while the path to the vending machine is connected. Since no better path to the vending machine is known, Contact Networking connects the IrDA path. A full path connection occurs:

- 1) The IrDA interface on the mobile node is bound to the link between it and the vending machine.
- 2) The mobile node establishes a connection to Contact Networking on the vending machine.
- 3) Contact Networking on the vending machine accepts the connection request and both nodes create local routes indicating that the channel between the two GRIPs is using this particular connection.

Upon establishing the connection Contact Networking forwards the queued data packets to the vending machine. Transaction processing then continues normally with no further involvement from Contact Networking. Prashant grabs his bag of Fritos™ and starts walking back to his office to write a paper about scatternet formation in Bluetooth. Within several seconds, the IrDA module on each machine informs Contact Networking that the connection has become idle. Contact Networking drops the idle connection and unbinds the IrDA interface since no other connections exist on the link.

2) *Local Communication with Handoff:* On his way to the office, Prashant passes a soda machine and decides to purchase a can of Coke™ to help wash down those salty Fritos™. The transaction proceeds similarly, until the impatient Prashant returns his mobile node to his pocket, interrupting the TCP flow over IrDA. The IrDA module in Prashant's mobile node informs Contact Networking of the premature connection breakage. Since the connection was being actively used by data traffic, Contact Networking immediately tries to find an alternate path to the soda machine in the network database.

Luckily for Prashant, the soda machine is also connected to an Ethernet segment bridged to the Wireless LAN. Contact Networking connects this alternate path. During the handoff procedure, no connection is available to support the channel to the soda machine, so any data packets are queued. After the neighbor handoff completes successfully, the channel is rerouted over the new connection and the queued packets are retransmitted. TCP on Prashant's node retransmits the single lost packet and recovers the transaction without the application's or the transport layer's awareness or participation. As in the previous example, the transaction completes normally and the connection becomes idle. Contact Networking at both ends drops the connection.

3) *Infrastructure Access:* On the way back to the office, Prashant decides to load Slashdot into Mozilla to check for interesting news—which can be challenging, with a bag of Fritos™ in one hand and a can of Coke™ in the other. CNS resolution of

`www.slashdot.org` fails since Prashant is nowhere near the Slashdot web server. When his DNS resolver tries to resolve `www.slashdot.org`, Contact Networking connects the path to the DNS server to complete the resolution, which returns an IP addresses for Slashdot's web server.

In this case, the first TCP packet of the flow is *not* destined for a neighbor in the network database, so Contact Networking connects its virtual path to the Internet:

- 1) Virtual path connection activates the underlying path to the Foreign Agent on the wireless LAN.
- 2) Once the Contact Networking connection to the Foreign Agent is established, MobileIP registration takes place.
- 3) When registration is complete, the virtual path is considered to be connected.

The TCP packet is finally forwarded over the newly established channel to Slashdot and Prashant's browser retrieves the web page normally. Several seconds after the page load completes, the virtual path becomes idle, forcing MobileIP deregistration and disconnection from the Foreign Agent.

6. Implementation

The prototype implementation of Contact Networking is developed on Linux atop a modified kernel. The implementation supports three different link types: infrared (IrDA), Ethernet and 802.11. The centerpiece of the implementation is the Connectivity Manager, a userspace daemon that runs on each Contact Networking node. Transparent application access to CNS is realized by an extension for the name service switch library.

A. Kernel Modifications

We extended the Linux 2.5 kernel by providing a mechanism for wireless interface drivers to export events to userspace programs. This extension became Wireless Extensions version 14 and has since been integrated into the development series Linux kernel at version 2.5.7, as well as back-ported into the stable kernel series at version 2.4.20. One event type we included in the Wireless Extensions is link-layer delivery failure notification. The drivers for Orinoco 802.11 cards (`orinoco_cs`), Aironet cards (`airo_cs`) and cards based on the Intel PrismII chipset (`hostap_cs`) were modified to send these notification events. These driver modifications are in various stages of integration into the kernel sources.

B. Connectivity Manager

In Contact Networking, each computer hosts a Connectivity Manager (CM) that configures its network interfaces and coordinates its discovery of and connection to neighbors, as well as maintaining the Contact Networking network database. The CM provides a common rendezvous point between user programs, the network stack and the routing tables.

1) Management and Adaptation Layers: The CM is organized as a process that reacts to asynchronous events. Low-level events are passed up from the link-layer specific adaptation modules, collectively denoted the adaptation layer, to the link-layer agnostic management layer. The management layer responds to events by directing the adaptation modules to connect or disconnect paths, or by modifying the network database. The management layer also responds to application name resolution queries from the Name Service Switch (NSS) interface (see Section 6-C), searching the network database for appropriate CNS records.

Adaptation modules signal the management layer when the following events occur:

- An interface is inserted/removed from the node
- A link is discovered/lost
- A neighbor is discovered/lost
- A path to some neighbor is discovered/lost
- A path has changed state
- A connection request was received from a neighbor

In response to these signals, the management layer uses an abstract API to manipulate interfaces, links, and paths. Each adaptation module provides a concrete implementation of the abstract API with behaviors appropriate to its link technology. The management layer uses these APIs to direct an adaptation module to bind/unbind an interface to a link, connect/disconnect a path on a bound link, or reject/accept a connection request from a neighbor by connecting the return path.

Contact Networking takes an approach different from that of traditional IP in providing network layer services. Contact Networking is aware of the characteristics of each link layer and uses link-layer specific functionality to provide better network service. Contact Networking has a link-layer specific adaptation module for each supported link type. Adaptation modules implement CNS querying and neighbor discovery in a lightweight manner, using native link-layer discovery mechanisms when available. Link-layer modules also inform the CM of dynamic interface insertion and removal or conditions that represent failure of a connection.

Link-layer modules also provide mechanisms to form connections to neighbors – path connection sets up bidirectional connectivity with the neighbor over the chosen path, binding an available interface to the path's link if necessary. Once a path is connected, the link-layer module monitors the connection for breakage with link-native mechanisms (e.g., failure to receive an acknowledgment on 802.11). Link-layer connection break indications cause the CM to reroute the channel to that neighbor over another path, if possible. Removal of a bound interface also causes rerouting to occur for any channels using that interface. The adaptation modules monitor connections for idleness and signal the manage-

ment layer to drop the connection. Disconnection tears down the neighbor connection, causing the interface to unbind from the link upon disconnecting the last connection. Unbinding and powering down the interface at the earliest possible time enables power conservation and allows the interface to be used to bind to other links.

All link-layer adaptation modules use a Linux `rt-netlink` socket to monitor interface creation events [28]. This enables each adaptation module to report interface discovery/loss events to the management layer.

The interface management aspect of Contact Networking is similar to Physical Media Independence (PMI), “an architecture for dynamically diverse network interface management” [14]. PMI observes the availability of network interfaces and informs applications of changes in interface state and availability, as well as coordinating a vertical handoff of the default route. Unlike the active interface management of Contact Networking, PMI passively reacts to network interface state changes.

2) *IrDA Adaptation Module:* The IrDA [17] adaptation module uses mechanisms native to IrDA and IrNET to provide services to the management layer. IrDA has a native discovery mechanism and a link-layer attribute exchange protocol (IrIAP). Path connection is realized by IrNET, which uses synchronous PPP for control, operating directly on IrDA frames. IrNET generates events to signal the IrDA adaptation module.

Upon receiving an IrNET discovery event, the IrDA adaptation module uses IrIAP to exchange CNS records with the new neighbor and delivers the resulting CNS record along with a neighbor discovery signal to the management layer. Since IrDA is a point-to-point technology, paths and links are unified in the IrDA adaptation layer. The architectural constraint of binding an interface to at most one link at a time mirrors the point-to-point nature of the link layer in Contact Networking. When the corresponding IrDA neighbor un-discovery event is reported from IrNET, the adaptation module reports both a lost path and a lost link event to the management layer.

Link block and connection request events are passed unmodified up to the management layer. The adaptation layer maps link-layer identifiers into the database entry for the path corresponding to the blocked connection or the requesting neighbor.

The IrDA adaptation module, when instructed to connect a path by the management layer, forks a child `pppd` on the IrNET control channel. The `pppd` “connect script” argument is actually directions for IrNET to choose which neighbor to connect. The binding completes when the IrNET event channel reports successful connection, which is signaled to the management layer. If PPP/IrNET connection fails, the `pppd` will exit prematurely and the IrDA adaptation module notifies connection failure to the management layer.

To force disconnection, the IrDA adaptation module simply kills the child `pppd`. When the connection has finally shut down, IrNET will report disconnection on its event channel, at which time the IrDA adaptation module marks the interface in the network database as once again available for binding to other links and reports disconnection to the management layer.

Forking a child `pppd` process for neighbor connection is a heavyweight procedure, so using IrIAP to exchange CNS records below IP provides low latency discovery, avoiding IP setup altogether. On a connection-oriented link layer like IrDA or Bluetooth, it is unclear how one would perform network-layer discovery. To which neighbor do you connect your link layer before sending, say, an ARP request? Contact Networking’s link-layer awareness enables support for discovery on connection-oriented link layers.

3) *Ethernet/802.11 Adaptation Module:* 802.11 [13], besides being wireless, is as different from IrDA as possible. 802.11 provides a simple datagram-delivery-port abstraction, with no additional frills built in. These facilities must then be directly coded in the 802.11 adaptation module. Interestingly, the additional code this creates is traded for code the IrDA module uses to monitor the external `pppd`. Both modules weigh in at almost exactly 2000 lines of C.

The 802.11 layer requires protocol support to handle many issues that IrNET/IrDA does not. Consequently, we devised a set of protocol commands that are encodable in a trivial subset of XML to facilitate easy communication between neighbor adaptation modules. This message encoding is used for CNS transport, path connection, path disconnection and idleness monitoring.

Lacking IrIAP, the 802.11 adaptation module must provide an alternate means to exchange CNS records. A UDP broadcast request/reply protocol enables the adaptation layer to query the CNS records of its neighbors on a link or reply to their CNS queries. Since CNS record exchange must precede any connection between two neighbors on the link, a CNS request necessarily contains an advertisement for the source. Neighbors use the CNS request/reply as an opportunity to record MAC addresses and GRIPs. The adaptation layer creates an ARP table entry for the neighbor’s GRIP with the discovered MAC address. The ARP entry’s lifetime is coupled to the CNS record, avoiding ARP resolution altogether for a neighbor whose MAC address is already known.

Neighbor CNS records discovered over 802.11 are immediately reported to the management layer. Unlike with IrDA, it is possible to discover many paths to different neighbors on the same 802.11 link. Path connection for 802.11 requires only a simple handshake of XML commands to ensure bidirectional connectivity, so connection is much lighter weight and lower latency than IrDA.

The 802.11 adaptation module monitors an `rt-netlink` socket for Wireless Event reports, specifically packet delivery failure notifications from the device driver. The device driver reports the unreachable neighbor's MAC address, which is used to locate the path corresponding to that neighbor in the network database. Currently, the prototype treats a single failure indication as path blockage and reports it as such to the management layer. Future work will analyze this decision to determine if some level of hysteresis may provide better performance than single-packet-loss path blockage.

Although connection establishment needs only a simple request/reply and a single route creation, connection monitoring for 802.11 is greatly complicated over IrDA. In IrDA, monitoring the interface's send/receive packet counters is sufficient to indicate idleness of link and path. In 802.11, it is possible to connect many paths over the same interface, so that it is impossible to determine *which* path or paths are idle by simply examining the packet send/receive counters. Our prototype uses the Linux iptables firewall support to add an iptables rule for each connection that only recognizes traffic on that connection. The adaptation layer can determine connection idleness by reading the iptables rule's packet counter.

The iptables rules created by the 802.11 adaptation layer are a rather problematic piece of state in that they can outlive the CM process. While under development and prone to crashing, the CM tends to litter the iptables with stale rules. To avoid this problem, the adaptation layer forks a child process that clears out all associated iptables rules when/if the CM exits unexpectedly.

Path disconnection is relatively simple. At the management layer's behest, or in response to a neighbor's disconnect request, the 802.11 adaptation layer destroys the corresponding route and iptables rule and sends a disconnect command to the neighbor. After the neighbor confirms disconnection, the final path disconnect notification can be reported to the management layer.

A future goal for the 802.11 adaptation module is to support access point scanning to determine the Extended Service Set Identifiers (ESSIDs) of nearby access points. Each distinct ESSID discovered can be modeled as a distinct link in the network database. Although we added access point scanning support to Wireless Extensions v14, time did not allow completely integrating this feature into the prototype.

4) Virtual Adaptation Module: This adaptation module exists to support infrastructure access via virtual paths to the virtual neighbor. To Contact Networking, the single virtual neighbor represents the entire Internet. In Figure 4, the mobile node has two paths available to the infrastructure through different providers. When informed of the discovery of a path P to a neighbor N that advertises infrastructure service, the management layer

directs the virtual adaptation module to create a corresponding virtual path P' with the virtual neighbor as its destination. The path P' is dependent on P , so that destruction of P causes destruction of P' . Semantically, this path dependency asserts that infrastructure access on the path P' is only possible while P is reachable. The dependency relationship between P' and P implicitly captures the fact that the node must direct packets destined for remote hosts through its neighbor N .

Requests from applications to contact non-neighbors are treated as requests to contact the virtual neighbor. When the user requires remote access, the normal path selection mechanisms in the management layer choose a virtual path P' . Connection of P' causes connection of the underlying real path P and initiates MobileIP registration through the neighbor N that P reaches, as well as creation of a default route with next hop N .

Virtual paths are monitored just like other paths. Therefore, vertical handoff for remote communication naturally falls out as an effect of neighbor handoff between virtual paths. When applications communicate with remote hosts, Contact Networking sees traffic over the virtual connection to the virtual neighbor. A break of the underlying connection is treated as a break of the virtual connection, which will cause the management layer to reroute the infrastructure channel over an alternate virtual path to some other infrastructure access provider.

5) Packet Handling: Contact Networking uses on-demand path connection and interface binding to provide maximum flexibility with minimum resources. To provide on-demand path connection, the CM requires a mechanism to intercept packets from local applications for neighbors to which it has valid paths. We call this mechanism the *demand channel*. Additionally, the demand channel must also be capable of capturing *any* non-local packets to support the virtual neighbor. Captured packets are queued until a path to the destination can be connected and then forwarded over the fresh connection.

The demand channel is implemented as a set of routes in the kernel routing table, one per neighbor, that points into a universal tunnel interface. The Linux universal tunnel is a device driver that attaches a network interface to a device file. Packets routed through the interface by the kernel may be read from the file and packets written to the file appear to have been received on the network interface. Packets thus captured are buffered in user-space.

Contact Networking also needs a mechanism to resend packets that it buffers during interface binding and path connection. Simply writing the packets into the universal tunnel will not suffice, since the mobile node is probably not configured to support packet forwarding. Our CM prototype uses a raw IP socket to inject unmodified captured packets into the network stack.

C. Application CNS Interface

The final piece to complete the Contact Networking architecture is a transparent application interface to CNS. Name resolution in Linux is handled through the Name Service Switch (NSS) library. NSS dispatches name/address queries to multiple name services, notably DNS and NIS. Contact Networking inserts a CNS name resolver into the NSS configuration, so that application queries try CNS before other naming services. Application queries are coded in XML and sent to the CM.

The name mapping component of the CM first tries to resolve queries from the network database. Failing that, any action taken to resolve the name is dependent on its structure. If the name ends in a suffix that identifies it as a link-layer specific alias—Currently `irda` for IrDA and `wlan` for 802.11—the name is passed to the appropriate adaptation module for on-demand resolution. The link agnostic suffix `adhoc` specifies that all adaptation modules should try to resolve the name on their links. Otherwise, the name mapping component returns a failure indication to NSS, which then continues to try to resolve the name with other name services as normal.

The wildcard alias `any` is handled by matching some appropriate neighbor entry in the network database. It is possible for naming conflicts to occur with wildcard aliases. For example, `any.irda` probably resolves to a single neighbor, but `any.wlan` or `any.adhoc` potentially aliases a large number of neighbors. When name ambiguity occurs, the name mapping component currently reports a “name not unique” error back to the original application. A more powerful mechanism for name disambiguation, possibly with user assistance, is a topic for future work.

7. Conclusions and Future Work

The main contribution of Contact Networking is to provide a complete solution for communication management on a local scale. Support for naming, routing, channel management and interface configuration in Contact Networking place localized networking on par with traditional Internet mobility. Contact Networking provides everything necessary for neighbor communication in the absence of the usual infrastructure service, while preserving ease of integration with the Internet when available. Unlike earlier work, Contact Networking is not dependent on the presence of the Internet for operation.

Contact Networking realizes the goals of a mobile node with multiple wireless interfaces through the application of link-layer awareness. Support for local communication differentiates Contact Networking from traditional mobility solutions that are infrastructure dependent. The Contact Naming Service provides on-demand naming and basic service discovery using link-layer na-

tive support. There are some important avenues we intend to explore to enhance Contact Networking.

Flexible Network Support for Mobile Hosts [35] extends MobileIP with support for multiple packet delivery methods, including MobileIP with or without reverse tunneling and/or route optimization, and regular IP. A mobile policy table enables selective mobility on a per-flows basis. Further, Flexible extends the MobileIP home agent to allow a mobile node to extend this flow-granularity routing support back to the home agent. The Flexible home agent enables different traffic flows to be routed to different care-of-addresses of the mobile node. In this way, Flexible enables mobile support for multiple interface remote communication.

Integration of Contact Networking and Flexible into a single system, along with a policy control to direct which traffic should use which interface, is an interesting avenue of future research. Allowing users to express preferences for network service to the CM would greatly increase the usability and applicability of the system.

Currently we are improving the interface management mechanisms present in Contact Networking. The approach we call co-link configuration will allow two devices to negotiate common link-layer binding configuration for shared interfaces. Co-link uses one interface to bootstrap others. For example, two PDAs could use IrDA to exchange a frequency and encoding key for secure 802.11 communication.

With no authentication, the local discovery and route management components of Contact Networking enable a local attacker to masquerade as an Internet host of the attacker’s choice. This vulnerability is a consequence of the absence of infrastructure support and can be seen as a magnification of the ARP security hole [2]. Peer-to-peer authentication approaches from ad hoc networking could address this issue in Contact Networking [12].

Other work has suggested transport protocols that can aggregate bandwidth from multiple interfaces simultaneously [11], [16]. Contact Networking provides a framework for discovery of multiple paths that these protocols can use. We are investigating the integration of Contact Networking with an aggregating transport protocol.

Many network services have static or infrastructure-dependent configuration. Notable among these services is DNS. A node is usually configured statically with a set of DNS servers. DNS, and similar services, could possibly benefit from the local service discovery model that CNS makes available. A mobile node that wishes to avoid the expense of remote communication could then use local DNS servers that it discovers through CNS. Generalization of this approach to other services with similar requirements would also be useful. Use of an IP-based protocol for service discovery, such as SLP [34], necessitates IP configuration before service discov-

ery may take place. An IP-based protocol cannot, therefore, be used to bootstrap IP.

Availability

We regret that we are unable to make our prototype available at this time. The current lack of documentation makes it nearly impossible for outside interests to install the prototype. We do intend to make Contact Networking available soon; interested parties should contact the first author at ccarter@uiuc.edu.

References

- [1] *AppleTalk Network System Overview*. Addison-Wesley Publishing Company, Inc., 1990.
- [2] S. Bellovin. Security problems in the TCP/IP protocol suite. *ACM Computer Communications Review*, 19(2), April 1989.
- [3] C. Brown and A. Malis. Multiprotocol interconnect over frame relay. Request for Comments (Standard) RFC 2427, Internet Engineering Task Force, September 1998.
- [4] S. Cheshire, B. Aboba, and E. Guttman. Dynamic configuration of IPv4 link-local addresses. Internet Draft (Work in Progress) `draft-ietf-zeroconf-ipv4-linklocal-07.txt`, Internet Engineering Task Force, August 2002.
- [5] R. Cole, D. Shur, and C. Villamizar. IP over ATM: A framework document. Request for Comments (Informational) RFC 1932, Internet Engineering Task Force, April 1996.
- [6] A. Conta and S. Deering. IPv6 stateless address autoconfiguration. Request for Comments (Draft Standard) RFC 2462, Internet Engineering Task Force, December 1998.
- [7] R. Droms. Dynamic host configuration protocol. Request for Comments (Draft Standard) RFC 2131, Internet Engineering Task Force, March 1997.
- [8] L. Esibov, B. Aboba, and D. Thaler. Linklocal multicast name resolution (LLMNR). Internet Draft (Work in Progress) `draft-ietf-dnsext-mdns-12.txt`, Internet Engineering Task Force, August 2002.
- [9] R. Hinden and S. Deering. Ip version 6 addressing architecture. Request for Comments (Standards Track) RFC 2373, Internet Engineering Task Force, July 1998.
- [10] C. Hornig. Standard for the transmission of IP datagrams over Ethernet networks. Request for Comments (Standard) RFC 894, Internet Engineering Task Force, April 1984.
- [11] H.-Y. Hsieh and R. Sivakumar. A transport layer approach for achieving aggregate bandwidths on multi-homed mobile hosts. In *ACM Mobicom '02*, pages 83–94, 2002.
- [12] J.-P. Hubaux, L. Buttyán, and S. Čapkun. The quest for security in mobile ad hoc networks. In *Proc. of the ACM Symposium on Mobile Ad Hoc Networking & Computing (MobiHOC '01)*, pages 146–155, October 2001.
- [13] IEEE Computer Society. 802.11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, June 1997.
- [14] J. Inouye, J. Binkley, and J. Walpole. Dynamic network reconfiguration support for mobile computers. In *Proceedings of the Third Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom '97)*, pages 13–22, September 1997.
- [15] D. Katz. Transmission of IP and ARP over FDDI networks. Request for Comments (Standard) RFC 1390, Internet Engineering Task Force, January 1993.
- [16] L. Magalhães and R. Kravets. Transport level mechanisms for bandwidth aggregation on mobile hosts. In *9th International Conference on Network Protocols ICNP 2001*, 2001.
- [17] P. J. Megowan, D. W. Susak, and C. D. Knutson. IrDA infrared communications: An overview. <http://www.irda.org>.
- [18] T. Narten, E. Nordmark, and W. Simpson. Neighbor discovery for IP version 6 (IPv6). Request for Comments (Standards Track) RFC 2461, Internet Engineering Task Force, December 1998.
- [19] NetBIOS Working Group. Protocol standard for a NetBIOS service on a TCP/UDP transport: Concepts and methods. Request for Comments (Standard) RFC 1001, Internet Engineering Task Force, March 1987.
- [20] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and R. W. Kevin. Agile application-aware adaptation for mobility. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles (SOSP) '97*, 1997.
- [21] C. Perkins. IP mobility support for IPv4. Request for Comments (Proposed Standard) RFC 3344, Internet Engineering Task Force, August 2002.
- [22] C. E. Perkins, J. T. Malinen, R. Wakikawa, and E. M. Belding-Royer. IP address autoconfiguration for ad hoc networks. Internet Draft (Work in Progress) `draft-perkins-manet-autoconf-01.txt`, Internet Engineering Task Force, November 2001.
- [23] D. Piscitello and J. Lawrence. Transmission of IP datagrams over the SMDS service. Request for Comments (Standard) RFC 1209, Internet Engineering Task Force, March 1991.
- [24] D. Plummer. Ethernet address resolution protocol: Or converting network protocol addresses to 48-bit Ethernet address for transmission on Ethernet hardware. Request for Comments (Standard) RFC 826, Internet Engineering Task Force, November 1982.
- [25] D. Provan. Transmitting IP traffic over ARCNET networks. Request for Comments (Standard) RFC 1201, Internet Engineering Task Force, February 1991.
- [26] J. Renwick. IP over HIPPI. Request for Comments (Draft Standard) RFC 2067, Internet Engineering Task Force, January 1997.
- [27] J. Romkey. Nonstandard for transmission of IP datagrams over serial lines: SLIP. Request for Comments (Standard) RFC 1055, Internet Engineering Task Force, June 1988.
- [28] `rtnetlink`, `NETLINK_ROUTE` - Linux IPv4 routing socket. Linux Man Page `rtnetlink(7)`, April 1999.
- [29] K. Sklower, B. Lloyd, G. McGregor, D. Carr, and T. Coradetti. The PPP multilink protocol (MP). Request for Comments (Draft Standard) RFC 1990, Internet Engineering Task Force, 1996.
- [30] A. Snoeren and H. Balakrishnan. An end-to-end approach to host mobility. In *ACM Mobicom '99*, 2000.
- [31] M. Stemm and R. H. Katz. Vertical handoffs in wireless overlay networks. *ACM Mobile Networking (MONET), Special Issue on Mobile Networking in the Internet*, 1998.
- [32] J. Tourrilhes and C. Carter. P-Handoff: A protocol for fine grained peer-to-peer vertical handoff. In *Proceedings of the 13th IEEE International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC '02)*, 2002.
- [33] J. Tourrilhes, L. Magalhães, and C. Carter. On-demand TCP: Transparent peer-to-peer TCP/IP over IrDA. In *Proceedings of the IEEE International Conference on Communications (ICC '02)*, 2002.
- [34] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. Service location protocol. Request for Comments (Standards Track) RFC 2165, Internet Engineering Task Force, June 1997.
- [35] X. Zhao, C. Castelluccia, and M. Baker. Flexible network support for mobility. In *Fourth ACM International Conference on Mobile Computing and Networking (MOBICOM '98)*, 1998.