Contact Networking: A Localized Mobility System

Casey Carter Robin Kravets Jean Tourrilhes
Department of Computer Science, UIUC *Hewlett Packard Labs*
{ccarter,rhk}@uiuc.edu jt@hpl.hp.com

***Abstract*— MobileIP, the standard for Internet mobility, enables transparent mobility for a mobile node, but requires communication to take a multihop path through the node's Home Agent. Although a user with a multiple-interface mobile node may desire the ability to communicate locally, perhaps while disconnected from the Internet, MobileIP offers no such support.**

Contact Networking provides lightweight, localized network communication to a node with diverse network interfaces, with emphasis on providing support for local connectivity equivalent to that provided by MobileIP for remote connectivity. The concept of link-layer awareness enables Contact Networking to tailor its operation to different links, using link-layer native services to implement abstract services when possible. Interface management and autoconfiguration insulate the user from concerns about the number and type of interfaces available.

In this paper, we motivate the need for localized mobility, and present the design and architecture of Contact Networking. Details of our prototype implementation illustrate the complexities of providing a localized mobility facility.

I. Introduction

Recent years have witnessed the development of a new class of mobile computing devices, merging personal communications and data access into a single device. One part mouse and two parts remote control, these mobile nodes are the interface point between users and the information environment. As users move through the world, they want to use their mobile nodes to maintain reachability through access to the global Internet, a form of mobility that is already well supported by MobileIP [20]. Users would also benefit from avoiding expensive multihop Internet connectivity and taking advantage of computing resources and services discovered locally, using whatever network interface is most convenient.

The inspiration for the name Contact Networking (CN) is to orchestrate direct communication between a mobile node and its neighbors. This focus on last-hop connectivity addresses the shortcomings of earlier work that assumes the presence of a network interface, with no concern for how that interface is configured and bound to the link (*binding* is the process of configuring an interface to use a particular link-layer medium). CN's approach to mobility support is diametrically opposite that of MobileIP. MobileIP treats all communication as though it is remote, even if local, whereas CN treats all communication as though it is local, even if remote. This difference in philosophy enables CN to provide lightweight service by being link-layer aware: tailoring itself to the characteristics of each link layer.

Providing localized network communication is more challenging when a node has multiple interfaces. Some link technologies are virtually plug-and-play and require almost no configuration, while others may require substantial configuration. Even in a simple neighbor communication scenario, coordinating through several interfaces with multiple neighbors simultaneously is challenging, given that those neighbors may support different non-overlapping wireless technologies. Diverse communication technologies either force users into tedious manual configuration to coordinate the interfaces, or restrict their access to a single interface at a time.

CN manages IP configuration and link-layer binding, using all interfaces to connect to neighbors. Exploration of the environment informs CN when communication is possible. As long as some pair of compatible interfaces exists, CN can maintain connectivity between neighbors. To preserve battery power and maximize flexibility, interface binding is delayed until applications indicate demand to communicate with a given neighbor. CN configures network interfaces to provide link-layer connectivity and bidirectional routing between the two nodes using a mechanism developed in earlier work [34]. Communication is maintained as links form and break by rerouting to other interfaces as necessary. This *neighbor handoff* mechanism (P-Handoff in earlier work [33]) is transparent to the user and ensures persistent connectivity.

In addition to active interface configuration and routing support, CN also provides name service and infrastructure connectivity. To enable operation when DNS service is unavailable, CN provides host and service discovery functions based on mechanisms native to each link layer. This Contact Naming System (CNS) allows users to refer to local resources with friendly names. Users often want to access services which are not available locally, for example, reading e-mail or purchasing a plane ticket. Since distant communication relies on the primitive hop-by-hop interaction of neighbors, CN extends to manage multihop end-to-end communications.

In this paper, we present the design of and case for CN. Section II motivates better mobility support for mobile nodes, particularly localized mobility. We present

This work was sponsored by Hewlett Packard Labs and National Science Foundation grant number ANI–0081308.

the requirements for localized networking in Section III, which is realized by the CN architecture of Section IV. Some examples detail the operational requirements for localized networking in Section V. Section VI discusses our prototype implementation of CN, including limitations and lessons learned. Section VII contains the natural progression from CN to future work.

II. Motivation

In this section, we define local and remote communication, and detail the lack of support in mobility solutions for localized networking with many wireless interfaces. We then advocate link-layer awareness, the idea that the network layer should incorporate link-layer knowledge to provide better network service. The following hypothetical scenario provides a context in which to discuss the issues of localized networking:

You and your colleague Xue board the train to travel to the client presentation. Xue hasn't seen the final version of the video, so you both point your PDAs' infrared ports at each other, and you push the video file to her PDA. Your PDAs connect over infrared, establishing a local communication channel without requiring Internet connectivity. Seconds after the transfer is initiated, you put your PDA back in your briefcase. This breaks the infrared link, so your PDA switches automatically to 5GHz radio to continue the transfer.

After a few moments, you decide to walk back to the dining car. Due to the limited range of 5GHz radio, that link also soon breaks, forcing the communication to fall back to 2.4GHz radio. As you walk away, you leave 2.4GHz radio range and your PDA's only option to preserve communication is to activate its GSM radio and communicate over the Internet with Xue's PDA. For the first time, your PDA registers with MobileIP and begins using multihop remote communication.

In the dining car, you use 5GHz radio to exchange business cards with a new acquaintance, without even needing to see his PDA – you simply push the card to whichever device in your vicinity responds to the nickname "Gerard." This transaction does not interfere with the ongoing transfer to Xue's PDA three train cars away.

As you return to Xue's car, the PDAs discover they can again connect to each other over local links. They shut down the expensive GSM radios and handoff the communication to the faster, cheaper 5GHz radio channel.

A. Local and Remote Communication

The mobile nodes in the scenario carry out two fundamentally different kinds of communication, which we term local and remote communication. Local communication is one-hop communication between a mobile node and another node with which it shares direct connectivity. The defining characteristic is locality—interacting with "this" device that the user discovers in the environment, such as a local printer. A local communication is a simple unplanned association between two hosts, requiring no prearranged network infrastructure access, but only that the two nodes share compatible communication hardware. Two users who meet each other and wish to exchange data between their mobile nodes, like the video in the scenario, use local communication.

Remote communication is between a node and some other explicitly addressed node in the network that is not in the immediate neighborhood. Web browsing is the classic example of remote communication. The web browser knows the web server's identity, but not its location. The mobile node must have access to the network infrastructure to communicate. Unlike the case of local communication, there is no possibility of spontaneous interaction. In the scenario, when the local communication links no longer suffice to communicate with Xue, the PDAs automatically fall back to remote GSM communication over the Internet.

Remote communication is more resource-intensive than local. A short local transaction could be completed by briefly powering up a network interface and exchanging packets. A remote communication requires a mobile node to power up an interface, find an infrastructure provider, register with its home agent, and then perform the communication before unregistering and powering the interface back down. The setup time necessary to register the mobile node's location, together with much longer delays incurred by the multihop path, keep the network interface powered over a longer time.

Current mobility approaches do not support a mobile node that requires concurrent local and remote communication. MobileIP focuses on multihop, remote communication [20]. MobileIP goes so far as to explicitly forbid a node to directly communicate with any neighbor except its Foreign Agent:

> While the mobile node is away from home, it MUST NOT transmit any broadcast ARP Request or ARP Reply messages. Finally, while the mobile node is away from home, it MUST NOT reply to ARP Requests in which the target IP address is its own home address. . .

In the scenario, had GSM access to the Internet been unavailable, the two PDAs would be unable to communicate via MobileIP even though they share compatible hardware. Without MobileIP, had one user brought along a network administrator, they could manually configure the devices. The administrator must choose:

- Use the shortest range, lowest power link supported by both nodes. This conserves battery power for the mobile nodes, but constrains the users' motion within the limited range provided.
- Use the longest range, highest power link supported

by both nodes. This provides a large area within which users can move freely. However, they pay for this flexibility with battery power.

Neither approach is optimal in all cases. From the users' perspective, the optimal solution would be to use the lowest power, shortest range links that provide connectivity at any given time. This dynamic approach is untenable for manual configuration.

B. Multiple Wireless Technologies

There is no perfect wireless technology for all applications: the design of a wireless technology is a tradeoff between range, capacity, and power consumption. As soon as a technology comes to market it is obsoleted by newer, faster, more efficient technology. There will always be periods of transition from the current to the new wireless technology, during which mobile nodes will support multiple links.

The management of multiple links is not integrated well into the traditional network stack. A multi-interface mobile node should be able to simultaneously utilize as many interfaces as necessary to satisfy its communication needs. In fact, users should not need to be aware of the number or type of interfaces available.

We refer to a less popular class of wireless technologies that have short range and directional transmission characteristics, as *Directional Area Networking* (e.g., infrared, laser). Although directional technologies are often deprecated, we see their limitations as features. Short range and directionality make directional technologies an ideal choice for a picking mechanism – selection of a particular device in the environment ("this" device) simply requires the user to point and click.

C. Link-layer Awareness

One of the best qualities of the Internet Protocol is its support for a variety of link layers by providing a uniform interface at the network layer. Although not every link layer was designed for IP, IP works over every link layer [3], [5], [10], [14], [22], [24], [25], [26], [27], [29]. This least-common-denominator approach loses link-layer specific advantages; by restricting its expectations to simple datagram delivery, IP loses the richness of the individual link layers.

Link-layer awareness provides access to the capabilities of each link layer. The advantages are 1) lightweight discovery mechanisms for finding new neighbors, 2) tighter attachment to neighbor nodes, allowing rapid detection of link failure, and 3) the ability to perform link-specific optimizations, such as header compression, when appropriate. The richness of wireless link layers in particular makes link-layer awareness more attractive.

To clarify, consider two different wireless link layers: IEEE 802.11 [31], and IrDA (Infrared Data Association) [16]. Both links natively detect link breakage. These link failure mechanisms provide useful information to a link-layer aware network layer: the aware network layer can immediately stop using the broken route and more rapidly find a new route. In the absence of link-layer failure indications, some periodic messaging system must be used to monitor link integrity. MobileIP monitors links between mobile nodes and foreign agents with periodic beacons. Link-layer awareness conserves resources—the link layer monitors the link, whether the network layer pays attention or not—and can provide more rapid notifications than higher-layer monitoring. In general, Link-layer awareness enables higher layers to transparently take advantage of implementations tailored to each link layer by providing abstract APIs for low-level services.

In the absence of link-layer awareness, the network layer must provide these basic services (e.g., neighbor discovery). The network layer is wasting resources by duplicating a service already provided in the link layer. In addition, the network layer cannot possibly provide service of the same quality provided by the link layer mechanism without tailoring its approach to specific link layer characteristics. The end result of this approach is a fragile service that is only suitable on a single link layer, precluding true interface heterogeneity.

III. Design Requirements

We present the services necessary to support local connectivity and configuration-free multi-interface networking. The user experience of localized networking must be identical to that of infrastructure networking: the user should not need to use different interfaces or tools for local vs. remote communication. We also address deployment, since a scheme to support localized networking is useless if not deployable in the current Internet.

A. Support Services

The critical services for localized networking are:

1) Neighbor Discovery
2) Name Resolution
3) On-demand Interface Binding
4) IP Autoconfiguration
5) Neighbor Routing
6) Channel Management
7) Infrastructure Access

We present a justification for each service requirement.

1) Neighbor Discovery: Neighbor discovery allows a mobile node to determine who its one-hop neighbors are. Neighbor knowledge enables the node to determine which interfaces may be used to contact a particular neighbor. In the absence of neighbor discovery service, a mobile node would be incapable of performing local communication at all.

IPv6 Neighbor Discovery (IPv6ND) [17] is unsuitable for providing discovery to mobile nodes. First, its op-

eration is not fully specified for multihomed hosts. Second, Neighbor Unreachability Detection may be too slow to support efficient handoffs when a neighbor is actively supporting a flow. Foremost, any IP-based solution obviously requires IP to be up and configured, conflicting with the goal of on-demand interface binding. Using IP as part of the mechanism to bootstrap IP creates a chicken-and-egg problem that is avoided by performing discovery at a lower layer.

2) Name Resolution: To support localized naming, a name-to-address mapping mechanism is necessary, like DNS provides in the Internet. If mobile nodes are to communicate directly, then each node must participate in the name-to-address mapping. Additionally, it is convenient for the user to select a nearby device without needing to know its proper name. For example, a name that denotes "the device my Directional Area Networking port is pointing at" enables the picking mechanism discussed in Section II-B.

3) On-demand Interface Binding: Binding is the process of configuring an interface to use a particular link-layer medium, like plugging an Ethernet card into a jack. Interfaces have an unbound state in which the interface may be able to perform some tasks, such as scanning for neighbors, but is not capable of full communication. Once bound, the interface can send and receive IP packets. Although they lack jacks, wireless interfaces also require configuration to select the link-layer medium. Even broadcast interfaces like 802.11 have parameters that bind the interface to one physical channel at a time (radio frequency, WEP key, and ESSID). To be link-layer neighbors, two nodes must have interfaces that are bound to the same medium.

There are two reasons to delay interface binding. First, since mobile nodes are battery powered, it is important to keep interfaces in a low-power state. Second, some interfaces provide point-to-point links, and can be used to communicate with only one neighbor at a time. To save power and allow flexibility in choosing to which link to bind, a local communication scheme must delay binding as long as possible. Therefore, interface binding should occur on-demand, when an application indicates demand to communicate with a neighbor. Certainly it is inefficient to maintain connectivity with neighbors with which the node has no need to communicate. Interface unbinding must also be automatic when demand is removed.

4) IP Autoconfiguration: Interface binding does little good if the user must manually configure IP address and netmask. To enable transparent network access, interface IP configuration (assigning IP addresses and netmasks to interfaces) must also be automatic.

Link-local (LL) addresses, in either IPv6 [9] or IPv4 [4], provide autoconfiguration. Unfortunately, addresses with scope restricted to a single link provide little support for persistent communication. These transient addresses cannot be used as endpoints for a transport layer connection, or that connection will fail upon moving. There is no method to map LL addresses to home addresses that identify mobile nodes. From the perspective of a mobile node, a LL address is little more than an IP-layer alias for the link-layer address. Further, both IPv4 and IPv6 address require an initial period of duplicate address detection (on the order of several seconds) before the address can be used: this delay is hardly conducive to performing rapid handoffs across multiple interfaces.

5) Neighbor Routing: The final step necessary to neighbor communication is the establishment of symmetric routing state. Neighbor routes are all that is needed to complete the local communication. Once these routes are available, application layer communication can continue normally.

6) Channel Management: These services provide local communication to the mobile node, but do not shield the user from managing links and interfaces. Two abstractions are fundamental to the operation of channel management. First, a *connection* is a link-layer abstraction that is formed between two neighbors who wish to communicate: it indicates that a handshake has taken place between these neighbors signifying that both have agreed to the communication. Second, a *channel* is the end-to-end abstraction through which transport flows propagate. We say that the channel between two nodes is realized by a particular connection at any given time.

Although connections can break due to mobility, the goal of mobility support is to preserve the channel across these breaks. If the user is actively communicating with a neighbor through a channel over a connection that fails, the mobile node must maintain the channel by connecting over another interface of possible (see Figure 1). Similarly, if a mobile node is connected to a neighbor over technology A, and a new, better path over technol-

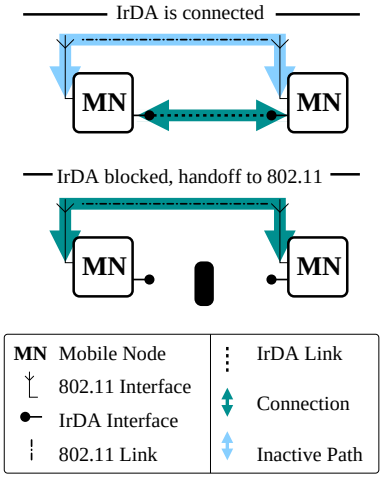

Fig. 1. Link-layer feedback and route failover

ogy B is discovered, the channel should automatically switch to the new interface.

7) Infrastructure (Internet) Access: Enabling a mobile node tocommunicate locally is an achievement, but not if the local focus completely occludes remote communication. The ability to access the Internet is far too critical to sacrifice in the name of other goals. In addition to the locally-oriented scheme, remote communication is necessary to complete localized networking support.

B. Deployment Considerations

The goals of autoconfiguration, neighbor routing, and channel management are not achievable by only a single node of a neighbor pair. Bidirectional communication necessarily requires support from both neighbors. Consequently, it is impossible to provide localized communication support when directly communicating with legacy network nodes. To be deployable, a localized communication scheme must minimize the impact on the installed base of hosts, and certainly must not require support from every node in a internetwork. To ensure future maintainability, the localized network facility should not violate proper network layering.

A localized scheme must put no requirements on applications. The design must allow unmodified binaries to work with existing networking APIs. One way to achieve this level of transparency is to isolate the scheme at the network layer or below, so that transport and application layers see the standard IP network-layer interface. Applications and users are presented with the illusion of continuous network service, with no effect other than a temporary aberration in service during handoff.

Application- and transport-layer mobility support advocates claim that applications must see mobility events in order to adapt to changed network service [19], [30]. Note that providing the illusion of network stability does not necessarily preclude the potential to provide mobility notifications to interested applications. It only reduces the impact on applications that are *not* mobility-aware.

IV. Architecture

In this section, we present CN's architecture and discuss how its individual modules fit the requirements presented in Section III. Since the requirements span from route management to interface configuration, CN lies at the junction between the link and Network layers in the OSI network stack, extending into both layers as depicted in Figure 2. CN includes a link-layer agnostic network layer module, and several link-layer aware modules. The components of the network layer module manage routing and select between multiple paths available to a neighbor through different interfaces.

Components of the link-layer-aware modules monitor the environment and report discovery events and link transitions. For local communication, link-layer modules monitor the link activity and inform the node when neighbors move out of range (i.e., a link breakage event). CN's link-layer discovery mechanisms incorporate the functionality of the Address Resolution Protocol (ARP) [23], mapping IP addresses to link-layer addresses.

Our architectural description of CN begins with a discussion of component structure, as depicted in Figure 2. The specific architectural components discussed are:

A. the network database, where the mobile node stores its model of the environment,
B. the IP autoconfiguration technique, which assigns addresses to bound interfaces,
C. the Contact Naming System (CNS), a name resolution scheme for CN nodes,
D. the CN approach to neighbor discovery,
E. the neighbor routing component,
F. the routing control, which directs interface binding, neighbor routing, channel management and infrastructure access,
G. the infrastructure access component, which CN uses to access the Internet.

For each architectural component, we describe its responsibilities in fulfilling the design requirements, as well as how it interacts with other components.

A. Network Database

CN includes a database that holds a model of the node's current view of the environment. This database is the central communication point for all parts of the architecture. The four basic abstractions with which it is populated are interfaces, links, neighbors, and paths.

- Interfaces are network attachment points on a node, such as Ethernet interfaces or cellular modems.
- Links are the communication media to which interfaces bind. An interface can bind to only one link

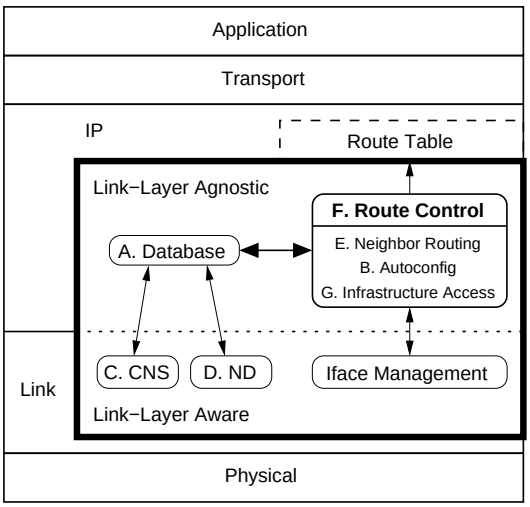

Fig. 2. Introducing Contact Networking into the OSI stack

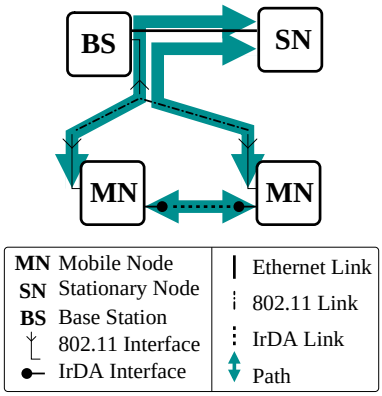

Fig. 3. Interfaces, Links, Paths, & Neighbors

at a time. Treating links as first-class entities makes it possible for the database to model both point-to-point (one neighbor per link) and broadcast (many neighbors per link) links.

- Neighbors are the nodes in the environment with which the user may wish to communicate.
- A path is essentially a pairing of a link and a neighbor, augmented with link-layer addressing information (the neighbor's address on that link).

Although an interface can bind to only one link at a time, CN can simultaneously establish connections with any and all neighbors to which it knows paths on that link. When CN wants to communicate with a neighbor, route control chooses a path to that neighbor from the database to connect. Path connection may necessitate interface binding (using the configuration state for the link in the database) and IP autoconfiguration. CN synchronizes path connection between neighbors, ensuring that both neighbors use symmetric routes.

A path can be in one of four states:

- Available - The path is believed to exist but is currently inactive.
- Connected - The path is activated and capable of transporting packets to the neighbor.
- Blocked - The path is in a transient unusable condition.
- Unblocked - The path has become usable again, but is not yet activated.

The blocked and unblocked states make it possible for CN to reroute traffic to another path temporarily (if an alternate is available) while the primary path is recovering.

To clarify these abstractions, Figure 3 depicts a network composed of two mobile nodes and one stationary node (the base station simply bridges the wired and wireless links, and is therefore not visible to CN). The stationary node has one interface, and each of the mobile nodes has two. There are two links in the figure, one IrDA and one 802.11/Ethernet (the two bridged segments form a single link). Each mobile node knows a path to the stationary node, and two paths (One over 802.11 and one over IrDA) to its mobile counterpart.

B. IP Autoconfiguration

A network interface must be configured with an IP address and subnet mask before IP can use it. In traditional IP, providing a unique address to each interface is a problem in its own right, requiring either administrative configuration or protocol solutions [4], [6], [7], [21]. These heavy-weight solutions require significant setup time to allocate an interface address, and create another problem when considering mobility: a transport flow cannot outlive the IP addresses used to identify its endpoints. If those addresses are transient, interface-specific addresses, then mobility is impossible. The goal of mobility is to provide a communication channel that persists beyond the lifetime of a single address-to-interface configuration.

CN embraces and extends the MobileIP approach to providing a persistent communication channel in the face of link mobility: use a single address (the *home address*) that is associated with the mobile node itself, and no subnet mask. MobileIP-based approaches configure the home address on a particular interface, providing mobile networking through only that interface, or selectively configure the home address on the single interface that currently has connectivity, enabling vertical handoffs [32]. CN extends this notion to its natural conclusion, and configures the home address on *all* network interfaces simultaneously.

Due to this expanded role of the home address in CN, we designate it the Globally Routable IP address (GRIP). Every node is identified by a unique GRIP. CN requires GRIPs to be permanently assigned, static home addresses, so that the presence and uniqueness of the GRIP can be guaranteed even when disconnected from the home agent/infrastructure. CN associates the GRIP of each neighbor with its entry in the network database. Use of the GRIP for local communication in CN neatly sidesteps the problem of providing a distinct IP address for each network interface. Configuring the GRIP on all interfaces allows the mobile node to persistently communicate through all interfaces simultaneously; the GRIP is guaranteed to live longer than any communication channel.

Using the same address on several interfaces is in direct conflict with the traditional IP addressing model in which one IP address denotes one network interface. Semantically, CN removes the emphasis that IP places on the role of the network interface. To CN, interfaces are transient entities, but the GRIP always stays the same. Since all application communication between nodes uses the GRIP, mobile nodes are accessible (and successfully mobile) as long as at least one network interface

is present. The choice of which interface to use for a particular channel becomes a pure routing problem (a choice of path from the network database) and not a question of interface management or configuration.

C. Name Resolution

The Contact Naming System (CNS) is CN's name resolution and name-to-address mapping facility. CNS names are structured exactly like DNS names, with textual names separated by dots. CNS names enable users and applications to transparently control the operation of CN by specifying CNS names in place of DNS names.

Every node has a full CNS name, which is exactly the fully-qualified DNS name that resolves to its GRIP. Agreement between CNS and DNS on the proper name for a node enables remote and local communication to use the same name.

CNS also supports link-layer specific name aliases, "wildcard" aliases, and service aliases that give a user flexibility in choosing a neighbor. Link-layer specific aliases combine a CNS name prefix with a suffix denoting a particular link layer. This facility enables a user to specify that they want, for example, `bob.irda`, which resolves to any neighbor whose CNS name starts with `bob` reachable via IrDA. The wildcard alias `any` (which may be qualified with a link-layer specific suffix) can be used to select any reachable neighbor. The infrared picking mechanism described in Section II-B can be easily implemented with the use of the alias `any.irda`.

A node replies to queries with its CNS record, containing its name and GRIP, and possibly other information (e.g., service information). For example, the node `bob.cs.uiuc.edu`, which provides print service and acts as a MobileIP foreign agent, might have the following CNS record:

GRIP: `128.174.244.37`
Name: `bob.cs.uiuc.edu`
Services: printer, MIPFA

CN attaches each CNS record to the corresponding neighbor record in the network database. Name queries that can be satisfied from the database require no network traffic. Link-specific alias queries that cannot be answered from the database only result in network traffic on links of the specified type. An application request to resolve `bob.irda` queries for `bob` on IrDA, but not 802.11.

CNS is similar to other multicast name resolution schemes [1], [8], [18]. The main difference is in the implementation. Our goal of maximal link-layer awareness implies that CNS may be transported by link-layer specific mechanisms instead of relying on IP multicasting support as in other approaches. Furthermore, unlike other schemes, the lifetime of CNS records is tightly coupled with neighbor knowledge in the network database, so long-term CNS caching is safe.

Each node monitors its interfaces for CNS queries requesting its configuration, although CNS may not use this request-reply model on every link layer. A CN node may use link-layer native mechanisms to retrieve CNS records. Since some links can perform discovery while in an unbound state, link-layer aware CNS conserves power and lowers delay over an approach that uses IP to resolve names. Optionally, nodes can actively advertise (either through a link-layer discovery method, or by periodic broadcast) their CNS records. These advertisements enable other interested nodes to easily collect CNS records by listening passively.

A node returns its CNS record in response to a the following types of CNS queries:

- Name queries
- GRIP queries, enabling reverse lookup
- Service queries (e.g., a node that provides MobileIP Foreign Agent service might respond to the name `MIPFA`)
- Wildcard queries (possibly link-layer qualified)

These CNS query types provide several ways for users to choose machines in their environment with which to interact, without requiring changes to IP network software that uses DNS names. CNS access is integrated alongside the normal DNS name resolution mechanism on the nodes, so that application name lookups try CNS resolution before DNS. A unified name resolution API makes localized networking easily available to network applications. For example, to purchase a can of Dr. Pepper™ from a vending machine a user would point the infrared port of their device at the vending machine and point their web browser at `http://any.irda`. The CNS resolver would return the vending machine's CNS record that specifies its name and GRIP. An exchange of business cards with another user named Bob would probably use the name `bob` to locate the GRIP of Bob's mobile node. We call this process "getting a GRIP" on the desired destination.

D. Neighbor Discovery

Neighbor discovery must incorporate link-layer awareness. Determining when another node is within the sphere of communication of a particular interface on a particular link is necessarily link-layer specific. Link-layer adaptation modules perform neighbor discovery and indicate neighbor appearance/disappearance to the link-layer agnostic module. Each link-layer aware discovery mechanism is tuned to the particular link, using native discovery mechanisms when available. This level of link-layer awareness provides high quality discovery on each link layer without compromising the generality of the rest of the CN system.

What other systems term neighbor discovery, finding a reachable peer on a link, is really path discovery in

CN. Actual neighbor discovery is only complete when two neighbors have exchanged CNS records, at which time CN updates the network database with a new path entry (this neighbor is reachable on this link at this link-layer address) and a new neighbor entry, if the neighbor is not already present (i.e., from being discovered on some other path).

CN's use of CNS records for neighbor discovery combines the neighbor discovery, address resolution, and name resolution into a single protocol. This combination reduces total overhead in contacting a neighbor, as well as allowing for long-term caching of name-to-IP address and IP-to-LL address mappings. The tight coupling of address mappings to neighbor lifetime means that mappings need never be refreshed, and are simply flushed when a neighbor goes away.

E. Neighbor Routing

CN takes link-layer network semantics, a flat address space with one-hop communication, and presents it to the transport layer as a set of host-specific routes, effectively providing an IP facade on link-local connectivity. This facade enables IP applications to use the localized networking facility equivalently to traditional IP support.

Routing necessarily affects the address model of CN. The use of GRIPs for local communication creates a class of *local* routes administered by CN. There is exactly one such route corresponding to each neighbor in the network database. A local route can be active, enabling packet forwarding directly to some interface of that neighbor, or inactive. An inactive route serves as an indication that CN can provide a path to that neighbor on demand.

Traditional IP routing, which views IP addresses as specifying a location within the internetwork, is retained in CN for routing to remote destinations. The result is a bipartite routing system that provides maximum flexibility in routing directly to neighbors—via local routes– but does not lose the ability to interface to the Internet with traditional routes. Integrating both styles of routing enables transitional deployment. CN mobile nodes can contact the current Internet through access points (much like MobileIP foreign agents) that provide CN service in addition to traditional IP routing. In the absence of CN-enabled access routers, a mobile node must resort to traditional techniques for unreachability detection and cannot provide rapid handoff when the link to the access router fails.

F. Route Control

Route control in CN handles on-demand interface binding and channel management. When a CN node first wishes to contact some IP address, it performs a lookup in the network database for a neighbor whose GRIP is that address. If the neighbor entry is present, its CNS record and at least one path will be known (the path over which the neighbor was discovered). Route control selects one of the available paths to connect, and negotiates the connection with the neighbor. If no record is present, the node falls back on traditional IP networking to find a remote route to the destination, possibly initiating infrastructure access as detailed in the next section. Route control also performs channel management by disconnecting idle connections (and unbinding idle interfaces when the last connection on that interface is dropped) and switching active channels to better paths when discovered.

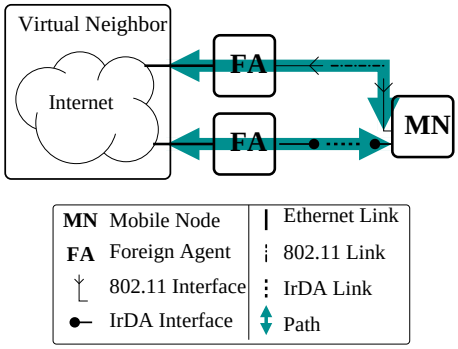

Fig. 4. Infrastructure as a virtual neighbor

G. Infrastructure Access

Since CN views all communication as direct neighbor communication, some extension is necessary to support remote communication. CN treats remote communication as local communication with a virtual neighbor that represents the entire infrastructure network. This conversion of remote to local communication enables CN to apply its normal mechanisms for on-demand establishment and channel management to infrastructure access.

In CN, a network node that wishes to provide infrastructure network access to others by performing as a MobileIP Foreign Agent advertises its willingness to do so through the Contact Naming System. Clients (mobile nodes that need infrastructure access) then discover the infrastructure provider normally through neighbor discovery. Upon discovering an access provider, CN creates a virtual path (path to the virtual neighbor) in the network database that corresponds to the path to the access provider (see Figure 4).

Integrating dial-up access into this architecture is trivial. A dial-up phone number is essentially a path description for a type of virtual path. Whenever a mobile node can use a dial-up interface, it is one hop away from the virtual neighbor. This virtual path would be statically configured into CN, with a simple monitor module that "discovers" the virtual path when a modem is plugged in or when within GSM coverage.

CN route lookups for a remote node (an IP address which does not correspond to a neighbor in the network

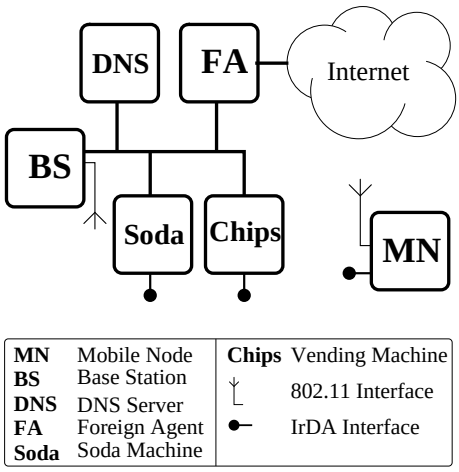

Fig. 5. Example network from Section V

database) are redirected to the virtual neighbor. Connecting a path to the virtual neighbor also connects its corresponding non-virtual path, and performs MobileIP registration to the access provider (i.e., the neighbor on the non-virtual path).

If a mobile node needs to maintain Internet access to enable externally initiated channels, CN can be configured to never drop virtual connections due to idleness. Normal neighbor handoff mechanisms would then preserve the channel to the Internet whenever possible, keeping the mobile node accessible to the rest of the world.

V. Examples

Some simple examples serve to illustrate the operation of CN. 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 CN 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, CN 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 application 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, CN 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 CN on the vending machine.
3) CN 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 CN forwards the queued data packets to the vending machine. Transaction processing then continues normally with no further involvement from CN. 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 CN that the connection has become idle. CN drops the idle connection, and (since no other connections exist on the link) unbinds the IrDA interface.

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 CN of the premature connection breakage. Since the connection was being actively used by data traffic, CN 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. CN 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 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. CN 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`, CN 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 CN 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 CN 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.

VI. Implementation

The prototype implementation of CN 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 CN 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 Intersil PrismII chipset (`hostap_cs`) were modified to send these notification events. These driver modifications are in various stages of integration into the mainline kernel sources.

B. Connectivity Manager

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

1) Management and Adaptation Layers: The Connectivity Manager is organized as a process that processes asynchronous events. Low-level events are passed up from the link-specific adaptation modules (collectively the adaptation layer) to the link 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 VI-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).

CN takes an approach different from that of traditional IP in providing network layer services. CN is aware of the characteristics of each link layer, and uses link-layer specific functionality to provide better network service. CN 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 Connectivity Manager of dynamic interface insertion and removal or conditions that represent failure of a connection (link-layer notification).

Link-layer modules also provide mechanisms to form connections to neighbors – path connection sets up bidi-

rectional 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 Connectivity Manager 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 management 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 `rtnetlink` 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 CN is similar to Physical Media Independence (PMI), "an architecture for dynamically diverse network interface management" [13]. 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 CN, PMI passively reacts to network interface state changes.

2) IrDA Adaptation Module: The IrDA [16] 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, the IrDA adaptation module simulates a link discovery for each neighbor. The end result is that there is a one-to-one correspondence between links and paths, so that the normal constraint of binding an interface to one link at a time ensures that the management layer will not try to connect to more than one neighbor at a time over IrDA. When the corresponding IrDA neighbor undiscovery 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 adaption layer maps link-layer identifiers into the database entry for the path corresponding to the blocked connection or 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 to 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? CN's link-layer awareness enables support for discovery on connection-oriented link layers.

3) Ethernet/802.11 Adaptation Module: 802.11 [31], besides being wireless, is as different from IrDA as possible. 802.11 (like Ethernet) provides a simple datagram-delivery-port abstraction, with no additional frills (such as native discovery and an event notification channel) 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 adaption 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 a CNS advertisement so that neighbors can 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 for IrDA.

The 802.11 adaptation module monitors an `rtnetlink` 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 (e.g., losing 2 packets in a row) 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 idleness of the connection simply 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 Connectivity Manager process. While under development and prone to crashing, the Connectivity Manager 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 Connectivity Manager exits unexpectedly.

Path disconnection is relatively simple. To disconnect (either 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 an XML 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 can be modeled as a distinct link entry 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 CN, 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 the management layer is informed of the discovery of a path P to a neighbor N that advertises infrastructure service, it 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 (non-virtual) path P and initiates MobileIP registration through the neighbor N to which P points, as well as creation of a traditional IP 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, CN 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: CN uses on-demand path connection and interface binding to provide maximum flexibility with minimum resources. To provide on-demand path connection, the Connectivity Manager 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 network stack) 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.

CN 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 Connectivity Manager 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 CN architecture is a mechanism to transparently interface applications 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. CN 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 Connectivity Manager.

The name mapping component of the Connectivity Manager 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.

VII. Conclusions & 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 CN place localized networking on par with traditional Internet mobility. CN 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, CN is not dependent on the presence of the Internet for operation.

CN realizes the goals of a mobile node with multiple wireless interfaces through the application of link-layer awareness. Support for local communication differentiates CN from traditional mobility solutions that are infrastructure dependent. The Contact Naming Service provides on-demand naming and basic service discovery using link-layer native support. There are some important avenues we intend to explore to enhance CN.

Flexible Network Support for Mobile Hosts [36] extends MobileIP with support for multiple packet delivery methods (MobileIP with or without reverse tunneling and/or route optimization, regular IP) to enable selective mobility. 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 CN 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 Connectivity Manager would greatly increase the usability and applicability of the system.

Currently we are improving the interface management mechanisms present in CN. 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.

In the absence of authentication, the local discovery and route management components of CN 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 CN [12].

Other work has suggested transport protocols that can aggregate bandwidth from multiple interfaces simultaneously [11], [15]. CN provides a framework for discovery of multiple paths that these protocols can easily use. We are currently investigating the integration of CN 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 [35], necessitates IP configuration before service discovery may take place. An IP-based protocol cannot, therefore, be used to bootstrap IP.

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] 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.

[14] D. Katz. Transmission of IP and ARP over FDDI networks. Request for Comments (Standard) RFC 1390, Internet Engineering Task Force, January 1993.

[15] L. Magalhães and R. Kravets. Transport level mechanisms for bandwidth aggregation on mobile hosts. In *9th International Conference on Network Protocols ICNP 2001*, 2001.

[16] P. J. Megowan, D. W. Susak, and C. D. Knutson. IrDA infrared communications: An overview. http://www.irda.org.

[17] 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.

[18] 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.

[19] 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.

[20] C. Perkins. IP mobility support for IPv4. Request for Comments (Proposed Standard) RFC 3344, Internet Engineering Task Force, August 2002.

[21] 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.

[22] 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.

[23] 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.

[24] D. Provan. Transmitting IP traffic over ARCNET networks. Request for Comments (Standard) RFC 1201, Internet Engineering Task Force, February 1991.

[25] T. Pusateri. IP multicast over token-ring local area networks. Request for Comments (Proposed Standard) RFC 1469, Internet Engineering Task Force, June 1993.

[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] I. C. Society. 802.11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, June 1997.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.