

Multi-Channel Multi-Interface Wireless Mesh Testbed

Chandrakanth Chereddi Pradeep Kyasanur Jungmin So Nitin H. Vaidya
University of Illinois at Urbana-Champaign

More details of our research at: <http://www.crhc.uiuc.edu/wireless>

I. MOTIVATION

Current day wireless technologies, such as 802.11a, support multiple non-overlapping channels. Multiple channels can be potentially used to increase network capacity by simultaneously communicating across different channels. In infrastructure based networks, multiple channels have been utilized by placing adjacent access points on different channels, thereby promoting simultaneous use of the channels. However, in ad-hoc (infrastructure-less) networks, a *single channel is typically used* for enabling adjacent nodes to directly communicate with each other.

Presently, most nodes are typically equipped with only one wireless interface. Recently, with reducing hardware costs, it is possible to equip nodes with more than one wireless interface. Nevertheless, *the number of interfaces at each node is expected to be smaller than the number of channels*. Our research [1]–[4] has shown that even in this scenario, network capacity can be significantly improved by using suitable protocols.

However, implementing these protocols in realistic testbeds is difficult, as the underlying operating system may not have the requisite architecture to support many of the features required by the protocols. For example, protocols may require interfaces to be switched among channels, but there is no architectural support for switching channels without modifying existing user applications. In this paper, we briefly discuss the challenges in supporting a multi-interface, multi-channel wireless network. We then present our architecture for implementing such a network, and describe a prototype implementation on a Linux testbed.

II. ARCHITECTURAL CHALLENGES

Our goal is to develop a generic architecture for supporting multi-channel multi-interface networks. We allow for the number of available channels to be larger than the number of interfaces.

Challenge 1: Implicit assumption that every neighbor uses a common channel

In most popular operating systems, the kernel maintains a routing table, with each tuple, containing a destination IP address and a corresponding interface (gateway) to use. However, when an interface is expected to send data over multiple channels (by switching), there is no in-built support for achieving this requirement. Presently, kernel routing tables assume that an interface will be capable of delivering data to a neighbor at all times, thereby making the implicit assumption *that all neighbors are on a single common channel*. This

assumption may not hold in multi-channel networks, since *different nodes may be listening to different channels*.

The wireless interface management APIs on most operating systems have support for changing channels with an explicit call from the userspace. For example, this can be achieved with the `iwconfig` tool on Linux. Yet, it is not possible to choose a particular channel based on the destination of the packet. Hence, a more fine-grained approach to channel control may be required to support *channel selection based on packet destination*.

Challenge 2: Interface and Channel Abstraction

Most applications today are written in an abstracted fashion using the socket API on top of the IP stack. If the complexity of having potentially multiple interfaces and multiple channels were not to be abstracted, many existing applications would have to be modified, which is undesirable. Also, if an interface is not on the desired channel, then the *applications need to carefully consider the current interface status* before actually switching to avoid packet loss.

If we wish to communicate with a neighboring node on a different channel, we need to carefully schedule the switching. For example, if we are presently on channel c and wish to send out a packet to a neighbor on a channel d , then the interface has to be scheduled to switch only after the communication on channel c is complete. Therefore, there is a need to provide *support for maintaining consistency between different switch requests*. Furthermore, when a packet cannot be immediately sent out on the desired channel, because the interface is then on a different channel, the packet has to be buffered for later transmission. Therefore, there is a need to *manage packet buffering, and scheduling packet transmissions*.

All these requirements argue for a separate module for abstracting the complexity of managing multiple channels and multiple interfaces.

Challenge 3: Supporting multi-channel broadcasts

Many applications rely on broadcast support (e.g., ARP, routing protocols). When all nodes are on a single channel c , broadcast is achieved by sending out a packet on channel c with destination address set to the interface broadcast address. Thus, broadcasting a packet ensures that *all neighbors of a node can receive the packet*. However, when different neighboring nodes listen to different channels, sending a packet out on a single channel may not be sufficient. One way to ensure all neighboring nodes can receive the packet is to *send a copy of the packet on every channel* by explicitly switching the interface to the every possible channel. In a

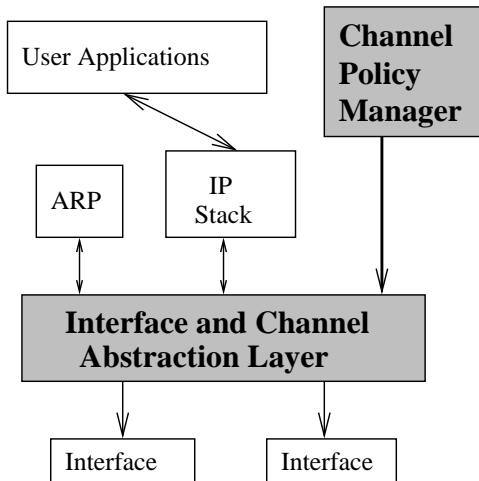


Fig. 1. Overview of the proposed architecture

multi-channel network, applications can explicitly send a copy of the packet on different channels, by explicitly switching the channel to be used by an interface. This again requires existing applications to be modified to be aware of the availability of multiple channels. Incorporating such modifications in existing applications requires non-trivial modifications, and breaks the layering principle as well (if the channel switching policy changes, then *all* applications that use broadcasting will have to be changed). Hence, there is a need to provide *support for broadcast such that it is transparent to higher-layer applications*.

The above challenges motivate our proposed architecture, shown in Figure 1. We develop a new module, called the “Interface and Channel Abstraction Layer”, that logically operates between the device drivers and the network layer (and ARP). This module provides support for managing multiple interfaces, interface switching, packet buffering, etc. The module exposes a *single virtual interface* to the higher layers. The module provides an interface for higher layers to specify the channel and interface to be used for each destination node. This information is stored in a table within the module, and the channel to be used for every packet is looked up in the table, based on the destination of the packet. The module maintains another table for supporting broadcasts that contains a list of channels on which a broadcast packet has to be sent out, and the list can be set up from the higher layers. The module sends a copy of each packet that has to be broadcasted on every channel in the broadcast list.

The tables in the module is populated by a higher-layer “Channel Policy Manager”. Different multi-channel protocols can be supported by substituting different policy managers. The main benefits of the proposed architecture are:

- 1) The architecture completely hides the complexity of managing multiple channels and interfaces from higher layers. Therefore, existing routing protocols, ARP, etc. can be used without any modifications.
- 2) The architecture is generic enough to support different protocols for assigning channels to nodes. These protocols can be developed entirely in the userspace as

customized “Channel Policy Managers”.

- 3) The architecture can also be extended to abstract multiple antennas, multiple channel rates, etc. from higher layer protocols.

III. IMPLEMENTATION OF THE PROPOSED ARCHITECTURE

We have implemented the proposed architecture in a Linux testbed. The present implementation of the “Interface and Channel Abstraction Layer” is written as an extension to the “bonding” driver in Linux kernel. Bonding driver in its unmodified form is capable of abstraction multiple interfaces and exposing it as a single interface to higher layer, and was designed for use in wired networks. We have extended the bonding driver to address all the challenges previously discussed.

Userspace daemons can inform the driver on the interfaces to be used for communication, which are then *bonded* together as a single interface. Next, through the use of new ioctl calls, the bonding driver can be informed of the channels each interface is capable of servicing. Later, routing tables, which contain tuples of IP address, interface and channel to be used, are populated through ioctl calls. Similarly, a broadcast list, which contains a list of channels and the interface to use for each channel, can be populated through ioctl calls.

Once the above information has been passed down to the kernel space, all existing network applications can be used unmodified. A channel policy manager daemon, which resides in userspace, deals with the complexity of channel assignments and updating routing tables when neighboring nodes switch channels, etc.

Channel switching is achieved through carefully designed timers, which are set based on a preset policy, to perform the channel switching and scheduling. Packets which are destined to a channel, other than the channel that is currently bound to an interface, are buffered. The buffered packets are later delivered to an interface for transmission when the interface has switched to the desired channel. All packets received by the wireless interface are sent to the higher layers bypassing the bonding driver.

As a prototypical example, we have implemented one channel policy manager that supports the *Hybrid Channel Policy assignment* we have proposed in [5]. The channel policy manager is responsible for discovering neighbors, and ensuring all the available channels are utilized.

REFERENCES

- [1] P. Kyasanur and N. H. Vaidya, “Capacity of Multi-Channel Wireless Networks: Impact of Number of Channels and Interfaces,” Tech. Rep., University of Illinois at Urbana-Champaign, March 2005.
- [2] J. So and N. H. Vaidya, “A Routing Protocol for Utilizing Multiple Channels in Multi-Hop Wireless Networks with a Single Transceiver,” Tech. Rep., University of Illinois at Urbana-Champaign, October 2004.
- [3] Pradeep Kyasanur and Nitin H. Vaidya, “Routing in multi-channel multi-interface ad hoc wireless networks,” Tech. Rep., University of Illinois at Urbana-Champaign, December 2004.
- [4] Jungmin So and Nitin H. Vaidya, “Routing and channel assignment in multi-channel multi-hop wireless networks with single-nic devices,” Tech. Rep., University of Illinois at Urbana-Champaign, December 2004.
- [5] P. Kyasanur and N. H. Vaidya, “Routing and Interface Assignment in Multi-Channel Multi-Interface Wireless Networks,” in *WCNC*, 2005.