MultiNet: Connecting to Multiple IEEE 802.11 Networks Using a Single Wireless Card

Paramvir Bahl   Pradeep Bahl   Ranveer Chandra

August 2003

Technical Report
MSR-TR-2003-46

There are a number of scenarios where it is desirable to have a wireless device connect to multiple networks simultaneously. Currently, this is possible only by using multiple wireless network cards in the device. Unfortunately, using multiple wireless cards causes excessive energy drain and consequent reduction of lifetime in battery operated devices. In this paper, we propose a software based approach, called MultiNet, that facilitates simultaneous connections to multiple networks by *virtualizing* a single wireless card. The wireless card is virtualized by introducing an intermediate layer below IP, which continuously switches the card across multiple networks. The goal of the switching algorithm is to be transparent to the user who sees her machine as being connected to multiple networks. We present the design, implementation, and performance of the MultiNet system. We analyze and evaluate buffering and switching algorithms in terms of delay and energy consumption. Our system has been operational for over twelve months, it is agnostic of the upper layer protocols, and works well over popular IEEE 802.11 wireless LAN cards.

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
`http://www.research.microsoft.com`

I. Introduction

There are several papers that articulate the benefits of virtualization [5], [9], [18]. However, to the best of our knowledge, the benefits of virtualizing a wireless card has been overlooked. In this paper, we propose MultiNet, a new virtualization architecture that abstracts a single wireless LAN (WLAN) card to appear as multiple virtual cards to the user. MultiNet allows these virtual cards to be simultaneously connected to physically different wireless networks. We describe our architecture in detail and then present a buffering protocol and two switching algorithms that give good performance for many common applications, such as telnet, ftp, file sharing and web downloads.

Our research is motivated by several compelling scenarios that are enabled with the above functionality. These scenarios include: increased connectivity for end users; increased range of the wireless network; bridging between infrastructure and ad hoc wireless networks, and painless secure access to sensitive resources. We discuss these in detail in Section II.

To enable these scenarios with current technology, one has to use a single WLAN card for each desired network. However, this is costly, cumbersome, and consumes energy resources that are often limited. An alternative to using more hardware is to use MultiNet and its accompanying protocols. MultiNet requires changes to the data link or device driver layer of the networking stack. It creates and manages multiple network stacks and maintains the associated state information for each network that the card is connected to. Simultaneous connectivity over all networks is achieved by switching the card between the desired networks and activating the corresponding stack. An advantage of this architecture is that it allows applications and protocols like TCP/IP to work without any changes.

In this paper we make the following four contributions:

- We present a new architecture for virtualizing WLAN cards and describe an overlooked connectivity paradigm that can be enabled using this architecture.
- We describe algorithms for switching between disparate networks and analyze their performance.
- We describe a buffering protocol that ensures packet delivery to switching nodes, and
- We describe and analyze a protocol for synchronizing nodes that switch between different networks.

As of this writing, MultiNet has been operational for over twelve months. During this time, we have refined the protocols and studied their performance. Many of the results we present are based on real working systems that include current and next generation IEEE 802.11 [12] wireless cards. For cases where it is not possible to study the property of the system without large scale deployment and optimized hardware, we carry out simulation based studies. Most of our simulations are driven by traffic traces that represent 'typical traffic'. For IEEE 802.11, our study shows that MultiNet nodes can save upto 50% of the energy consumed over nodes with two cards, while providing similar functionality. We also quantify the delay versus energy tradeoff for switching nodes over performance sensitive applications. Although we have built MultiNet over IEEE 802.11 wireless LANs, our approach is not limited to this standard.

The rest of this paper is organized as follows. In Section II we present some scenarios and applications that motivated us towards building MultiNet and for which MultiNet is currently being used. Section III provides the background needed for the rest of the paper and Section IV presents some related research. The MultiNet architecture is presented in Section V, and its implementation is described in Section VI. Performance and feasibility are discussed in Section VII and VIII. Future work is presented in Section IX and we conclude in Section X.

II. Motivating Scenarios

The MultiNet virtualization architecture enables several new applications that were earlier not possible using a single wireless card. We enumerate a few of these here.

- *Concurrent Connectivity:* A user can connect her machine to an ad hoc network, while staying on her authorized infrastructure network. For example, consider the case where *Kisco's* employees conduct a business meeting with *Macrosoft's* employees at Macrosoft's headquarters. With MultiNet and a single wireless network card, Kisco employees can share documents, presentations, and data with Macrosoft's employees over an ad hoc network. Macrosoft's employees can stay connected to their internal network via the access point infrastructure while sharing electronic information with Kisco's employees. Macrosoft does not have to give Kisco employees access in their internal network in order for the two parties to communicate.

- *Network Elasticity:* The range of an infrastructure network can be extended by allowing border nodes to act as relays for authorized nodes that are outside the range of the Access Point (AP). For example, a node *X*, associated to a home AP, is being used to browse the web. Another node *Y* is moving while connected to the same AP and looses its connection because it goes out of range. With MultiNet, if *X* is within range of Y, it can connect to *Y* over an ad hoc network, and forward Y's traffic on to the AP.
- *Gateway Node:* A node that is part of a wireless ad hoc network and close to an AP, connected to the Internet, can become a gateway node for the ad hoc network [8]. This node becomes a bridge for other nodes on the ad hoc network, passing their packets to and from the Internet.
- *Network Security:* Different groups (e.g. human resources personnel, secretaries, developers etc.) within a company may be given different permissions to access data servers. These servers could be on physically different networks. For a privileged user, who has permission to access different networks, having MultiNet is valuable. She would not have to disconnect and reconnect between networks every time she wishes to access resources on different networks.
- *Increased Capacity:* The capacity of ad hoc networks can be increased when nodes within interference range can communicate by switching on orthogonal channels [15].
- *Virtual Machines:* Users can connect different virtual machines [9], [18] to physically different wireless networks.

All the above scenarios require nodes to be connected to more than one wireless network. In the sections that follow, we discuss the pros and cons of MultiNet and show why we consider it the preferred alternative in terms of cost, physical space, and improvement in battery lifetime.

III. Background

We first discuss the limitations of IEEE 802.11 networks and describe why maintaining simultaneous connections to multiple wireless networks is a non-trivial problem. We then briefly describe the next generation of WLAN cards, over which we evaluate MultiNet.

A. Limitations in Existing Systems

Popular wireless networks, such as IEEE 802.11, work by association. Once associated to a particular network, either an AP based or an ad hoc network, the wireless card can receive and send traffic only on that network. The card cannot interact with nodes in another network if the nodes are operating on a different frequency channel. Further, a node in an ad hoc network cannot interact with a node in the infrastructure network even when they are operating on the same channel. This is because the IEEE 802.11 standard defines different protocols for communication in the two modes and it does not address the difficult issue of synchronization between different networks. As a matter of practical concern, most commercially available WLAN cards trigger a firmware reset each time the mode is changed from infrastructure to ad hoc or vice versa.

B. Next Generation of IEEE 802.11 WLAN cards

In order to reduce the cost and commoditize wireless cards, IEEE 802.11 WLAN card vendors [1], [4] are minimizing the functionality of the code residing in the microcontroller of their cards. These next generation of wireless cards, which we refer to as NextGen 802.11 cards, implement just the basic time-critical MAC functions, while leaving their control and configuration to the operating system. More importantly, *these cards allow the operating system to maintain state and do not undergo a firmware reset on changing the mode of the wireless card.* This is in contrast to the existing cards, which we refer to as legacy wireless cards in the rest of this paper.

IV. Related Work

To the best of our knowledge, the idea of simultaneously connecting to multiple wireless networks has not been studied before in the context of wireless LANs. A related problem was considered for scatternet formation in Bluetooth networks in [13], [14] among others. Bluetooth networks comprise basic units, called piconets, that can have at most 7 nodes. Piconets are used to form bigger networks, called scatternets, by having some nodes on multiple piconets. However, the problem of enabling nodes in Bluetooth networks to listen to multiple piconets is significantly different from the problem of allowing nodes to connect to multiple IEEE 802.11 networks. Bluetooth uses a frequency hopping scheme for communication between multiple nodes on the network. A node can be on two networks simultaneously if it knows the correct hopping sequence of the two networks and hops fast enough. IEEE 802.11 networks, on the other hand, have no such scheme as we described in Section III.

The concept of virtualization has been studied in the context of operating systems. VMware [5] uses virtualization to run multiple operating systems on a single computer. Disco [9] is a virtual machine monitor (VMM) that runs multiple operating systems on a scalable multiprocessor and the Denali [18] Isolation kernel uses virtual machines (VMs) to host multiple untrusted services. These systems virtualize all the hardware resources, including the network interface, to form multiple virtual machines. Each virtual machine has its own virtual network interface that it uses for sending and receiving packets.

There are significant differences in the motivation and design of MultiNet and the above virtualization schemes. While VMMs are designed to support multiple operating systems through VMs on a single machine, MultiNet is aimed at supporting multiple networks over one WLAN card. The VMM approach does not work with IEEE 802.11 WLAN cards. VMMs have a bridged architecture in which virtual NICs have MAC addresses that are different from the MAC address of the underlying wireless card. There is no provision for this approach in the IEEE 802.11 standard. Further, different virtual NICs in VMMs may require connectivity to physically different wireless networks. This is currently not possible using existing virtualization schemes.

V. The MultiNet Approach

MultiNet defines virtualization of a wireless card as *an abstraction of multiple wireless networks as different always active virtual adapters over a single WLAN card.* It should be possible for a user to change individual parameters of each virtual adapter, and these always active adapters should also be able to send and receive packets at any time. MultiNet achieves this by multiplexing the wireless card across multiple networks. It uses an adaptive network hopping scheme where a card gets a time slot, called the *Activity Period,* for each network operating on a particular channel. The sum of the activity periods over all the connected networks is called the *Switching Cycle*. We describe MultiNet within the context of the following four questions:

- What changes do we need to make in the networking stack to support MultiNet?
- What buffering protocols should we use to ensure delivery of packets across the disjoint networks?
- What switching algorithms should we use to get the best performance from MultiNet? and
- How do we do synchronize multiple switching nodes in an ad hoc network?

We limit the scope of this paper to the study of single-hop networks only, which may be infrastructure-based or ad hoc. Several additional interesting questions arise when considering MultiNet over multihop networks, but we do not consider these in this paper and leave them for future work.

A. The Virtualization Architecture

The virtualization of wireless adapters is implemented by the *MultiNet Protocol Driver* that is placed as an intermediate layer, between IP and the MAC. This driver exposes the wireless LAN media adapter as multiple always active virtual wireless LAN media adapters, one per desired network. The IP stack sees all the adapters as always active even though at the driver level only one is active (connected) at any given time. Other than exposing virtual adapters, the MultiNet Protocol Driver is also responsible for switching the wireless cards across the different networks, and buffering packets for networks that are currently inactive.

Figure 1 illustrates the virtualization of a WLAN card when a user wants to connect to 3 wireless networks. The MultiNet Protocol Driver exposes 3 virtual adapters, and all of them appear active to IP although only Network 2 is active at the instant shown in the figure.

B. Buffering Protocol

The upper layers see the virtual adapters as active even though they may be passive at the driver level. The MultiNet Protocol Driver handles application sends and receives over all the connected networks using a simple buffering protocol. We describe this protocol in the subsections below.

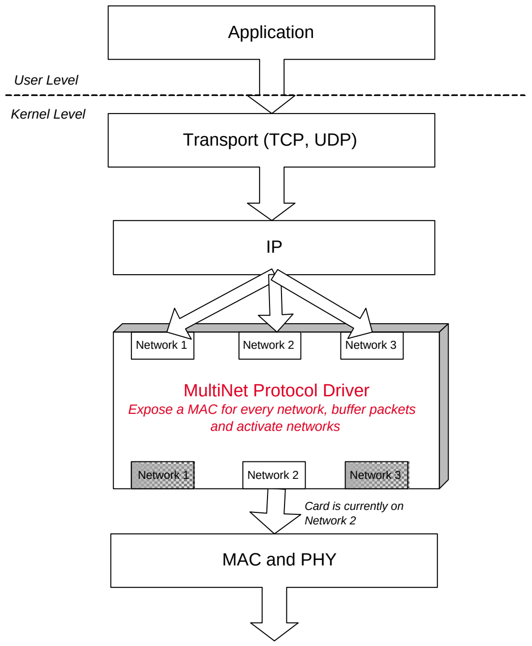


Fig. 1. Modified network stack

1) Packets Sent from the MultiNet Card: When the upper layers send a packet to the MultiNet virtual adapter, the MultiNet Protocol Driver does either of the following:

- If the adapter is active at the driver level, the packet is sent to the card for transmission.
- If the adapter is passive at the driver level, the driver buffers the packet.

The buffered packets are sent as soon as the corresponding virtual adapter is activated. The maximum queuing delay occurs when a packet arrives on a virtual adapter just after it has been inactivated. In this case, the driver waits for the entire switching cycle before it can send the packet to the card. This delay for a packet on network i can be formulated as $\sum_{j \neq i}(T_j + \delta_j) + \delta_i$, where T_j is the activity period of network j and δ_j is the time taken to switch to network j.

2) Packets Sent to the MultiNet Card: Packets sent to a switching card over network i will be lost if the card is in a different network j at that instant. In addition to sends, the buffering protocol has to ensure delivery of all packets sent to a MultiNet node. This requires nodes to buffer packets for MultiNet nodes that are currently unavailable, but which will become available in the near future.

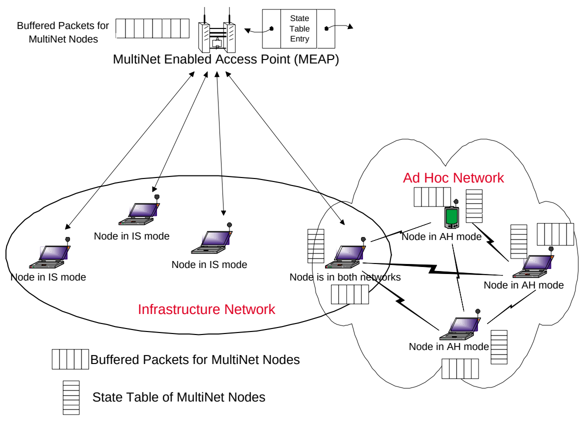


Fig. 2. Buffering in MultiNet

The MultiNet buffering scheme is illustrated in Figure 2. This scheme defines different behavior for APs and nodes. APs maintain the state of all the cards that are associated to it. Nodes instead maintain the state of directly reachable cards in any ad hoc network to which they are connected. The state table is a 4-tuple state space, containing information about the card's address, its Service Set Identifier (SSID) [12], the time when the card switched from this SSID, and when it is expected to switch back to this network.

The state table mentioned above is updated as follows. A card sends a packet to the AP in an infrastructure network, and to all nodes in an ad hoc network, just before switching from it. This packet informs all the reachable nodes in the current network of its temporary unavailability. The packet also contains information about the duration of absence from the current network and is used to maintain the state table at the nodes and APs. So, whenever a node or an AP has a packet to send to another node, it will first check the state table. If the destination card is currently associated to another network, the packet is buffered. When the state table indicates that the timer of a card has expired, and that it would have switched back to a network, the nodes and APs send the packets buffered for that destination on the corresponding network.

A concern for the buffering protocol is the loss of broadcast packets carrying state in the ad hoc network. Fortunately, as we will describe in Section V-D, switching cards in ad hoc networks send more than one packet carrying this state per switching cycle. To handle situations when none of these messages get through, neighbors temporarily buffer packets for the switching node if it is unreachable. They estimate the activity period of the switching node based on past activity, and keep probing it. On receiving a response, they transmit the buffered packets.

3) Properties: The buffering protocol is based on the assumption that nodes maintain their promises, and that they do not lie about when they expect to get back to a network. This is a reasonable assumption because there is no incentive for a node to lie. A lie will only hurt the liar since an incorrect time estimate will lead to degraded performance. A value larger than the actual switching time causes an extra delay in getting packets, while a smaller value results in packet loss. Therefore, the nodes can be expected to stay honest and transmit the right switching period.

The above protocol also ensures transparent switching: applications using a switching card do not have to know its state. Packets sent to the switching card are all received, though with some extra delay, as long as everybody is honest about their switching period.

4) Buffering on IEEE 802.11 Access Points: The MultiNet buffering algorithm for infrastructure networks can be implemented without changing the software or hardware at the APs. **The trick is to use the Power Saving Mode (PSM) feature available in IEEE 802.11 networks.** So, the MultiNet cards will fake PSM to the APs when they switch to another network. When a card enters PSM, the AP automatically buffers packets for that card. Although the AP thinks that the card is asleep, the card actually switches and is actively connected to other networks. After the sleep interval, the card connects to the AP and receives all the buffered packets.

C. Switching Algorithms

MultiNet services packets on all networks by staying on each of them for a certain Activity Period before switching to the next network. We describe the switching algorithms below.

1) Strategies: We propose two strategies to determine the activity period of each network.

- **Fixed Priority:** Each network is assigned a fixed pre-allocated Activity Period. This time is prioritized, with some networks getting more time than the others. These priorities are specified by the user based on his or her requirements.
- **Adaptive Schemes:** The Activity Periods of the networks are decided adaptively based on the amount of traffic seen by the node on the network. A more active network gets more time as compared to a less active one. So, if MultiNet has to switch across N networks, and network i has seen P_i packets in its last Activity Period ATP_i, then the node stays in network j for an Activity Period given by $(P_j/ATP_j) * (1/(\sum_{\forall k} P_k/ATP_k)) * (\sum_{\forall k} ATP_k)$. The first term gives the network utilization of network j, the second gives the utilization across all networks, and the final term is the total amount of time the node is active across all networks. The adaptive strategies give a default time to a network that has not transmitted any traffic. This is necessary to handle the case where a node just wants to receive packets on a network. We study two different adaptive strategies for MultiNet. **Adaptive Buffer** adapts the Activity Period to the number of packets buffered on each network, while **Adaptive Traffic** adapts to the number of packets sent and received on each network over a predefined time window.

2) Switching on IEEE 802.11 Network Cards: Implementing the above strategies using IEEE 802.11 PSM requires some modifications. In PSM the node sleeps for some number of AP Beacon Periods, called the *Listen Interval*. Consequently, the granularity of the Activity Periods on each infrastructure network is the AP Beacon Period. The Listen Interval is rounded to the multiple of the Beacon Period that is closest to the Switching Cycle. We also allow MultiNet nodes to sleep to save power. The nodes follow a protocol similar to the IEEE 802.11 PSM. If it does not have any packets to send on a particular network, it goes to sleep for the rest of its Activity Period. It then wakes up after the Activity Period to be active on the next network.

D. Synchronization Protocol

While the protocol described in Section V-B works for multiple cards switching between infrastructure networks, it has to be enhanced to handle multiple cards switching in and out of an ad hoc network. We first describe the scenario in infrastructure and ad hoc networks, and then present a distributed algorithm to solve the problem.

1) Effect on Infrastructure and Ad Hoc Networks: In infrastructure networks the AP stays on the same network all the time. This property ensures that packets transmitted from cards connected to an infrastructure network always get through, and packets to these cards are buffered at the APs if the card is currently not in that network (or sleeping). The performance of the network is affected only by the buffering capability of the AP and the switching card. It is unaffected by the number of switching cards. For all the cases that we studied, the end-user performance is always good with packets sent by or to the switching cards always getting through.

The situation is different for ad hoc networks, and we describe the problem using the following scenario. Suppose Bob and Alice are co-workers who belong to the same organization. Further, suppose Bob is sharing a presentation with Alice over an ad hoc network. Both are also connected to the corporate infrastructure network using MultiNet. A worst case arises when both Bob and Alice switch synchronously to the other network. That is, Bob switches to the infrastructure network just when Alice switches to the ad hoc network, and vice versa. As a result Bob and Alice are never in the same network at the same time, and therefore will never be able to communicate. Note that Bob and Alice might be able to communicate over the ad hoc network if there is a another person, say Trudy, within the range of both Alice and Bob, whose time in the ad hoc network overlaps with the duration of both Bob and Alice. However, there might be situations where Trudy is not present. Our switching protocol is designed to handle such scenarios.

2) A Distributed Switching Algorithm: The key to handling multiple switching cards is synchronizing the time during which cards stay in an ad hoc network. To enable cards in an ad hoc network to communicate with any other node in that network, there must exist an overlap in the time periods for which the members stay connected to the ad hoc network. We present a simple and efficient distributed algorithm that ensures concurrent connectivity among cards in an ad hoc network for at least a brief time period every switching cycle. The algorithm aims to converge the switching times for all the cards in an ad hoc network. Therefore, it tries to achieve synchronized switching to and from the ad hoc network for all members of that network. This solution is valid for single hop ad hoc networks. We do not attempt to solve the problem for multihop networks.

Notation: We use the following variables for describing our algorithm:

- ATP_i: Activity (Time) Period in network i. This is the duration the node stays connected to network i.
- δ_i: The time taken to switch to network i. This depends on the overhead of the switching protocol.
- SC: Switching Cycle. This is the time interval for cycling through all networks. It is a constant fixed by the MultiNet software, and is known to all the nodes. It is the aggregation of the activity time periods and switching delay of all the networks, along with the sleep time, if any.
- $TEATP_i$: This is the time elapsed inside ATP_i. It can be anywhere from 0 to ATP_i.

Algorithm Description: When a MultiNet node switches to a network j where it has not yet formed an ad hoc network with any other node, it stays on the network for at least 2*SC time to hear announcements from other nodes in j. It also announces itself in the beginning of this interval. Thereafter the MultiNet node announces itself every SC time interval in network j. An announcement carries information about ATP_j and $TEATP_j$ for the announcer. Nodes use this announcement to synchronize their time with the leader of the ad hoc group

We define a leader of an ad hoc network to be a node with the largest MAC address in that ad hoc network. So, if a MultiNet node hears an announcement from another MultiNet node with a bigger MAC address, it marks the announcer as its leader and changes its ATP and $TEATP$ to that of the leader. Since the node knows the SC, which is the same for all nodes, it is able to synchronize with the leader. It also stores the MAC address of the leader so that if the leader changes in the future, it can resynchronize itself with a new leader. If the announcement is from another MultiNet node with a smaller MAC address, the receiving node announces its ATP and $TEATP$ after a random time interval in the range of 0 to *WaitTime*. *WaitTime* is less than or equal to minimum of the amount of time left to finish the network active time period, i.e $ATP_j - TEATP_j$, of the node that triggered the announcement and the announcer. If nodes on a network stop hearing the leader for 2*SC time interval i.e. for two full switching cycles, they assume that it is gone and resynchronize to the next leader.

Discussion: The above protocol aims at providing randomized synchronization among all the ad hoc nodes, with all the nodes synchronously switching into and out of the ad hoc network with a high probability. Moreover, in the absence of faulty nodes and when no announcement messages are lost, if the MultiNet nodes in the ad hoc network j have the same ATP_j value, then new MultiNet nodes joining network j will converge on a single ATP_j after a period of time no longer than $2 * SC$. This is true since a new node joining the network has to wait at most $2 * SC$ time to hear an announcement from another node in the ad hoc network j. This node synchronizes itself with the ad hoc network in the first SC time period, and uses the remaining time to determine the value of ATP_j.

An area of concern is that announcement messages are broadcast and are therefore unreliable. So it is possible for the announcement message to not reach all the ad hoc nodes. There might exist nodes that do not receive any announcement messages in a SC time period. Fortunately, once a node has successfully joined an ad hoc network j and assigned a value to ATP$_j$, it mainly uses the announcement messages to maintain synchronization to the ad hoc network. So, MultiNet is not drastically affected by the loss of a few announcement messages.

The bigger problem with the above algorithm occurs when MultiNet nodes want to be connected to more than one ad hoc network. It is possible that nodes with largest MAC addresses in the two networks have overlapping times of activity in each network. In such a case, a node cannot be connected to both the networks, and even if it does, it may not synchronize its ATP values. We consider two solutions to solve this problem. The node might try to arbitrate with the leader of each ad hoc network to have non-overlapping periods of activity in each network. This approach has repercussions on other nodes in the network, and we therefore use the second solution in which the node joins both networks, but synchronizes its ATP value to only one network based on its priority. It remains connected to the other network for the remaining duration, but does not send any announcement messages with its ATP in this network. This ensures connectivity to both the networks. However, the node is not permitted to connect to both the networks if their activity periods overlap completely.

A disadvantage of this protocol is that it prohibits adapting switching durations for ad hoc networks based on local decisions at MultiNet nodes. In our current design, we make these decisions at the leader node, and the other ad hoc nodes synchronize themselves based on this value. It should be noted that this is just one solution and we are currently looking at better algorithms for enabling adaptability with synchronization in an ad hoc network. However, the above protocol works for purposes of MultiNet, and a better algorithm will only improve the results presented in Section VII. We note that the above protocol coupled with the Buffering Protocol, described in Section V-B, only requires loose synchronization among nodes to ensure reliable packet sends and receives. While the switching protocol makes sure that any two nodes in an ad hoc network are connected for some time, the buffering protocol handles the sends and receives when nodes switch between multiple networks.

E. State Diagram of a MultiNet node

The working of MultiNet is illustrated using a state diagram in Figure 3 comprising eight states. It is assumed that the wireless card running MultiNet is connected to a maximum of n networks, $\{N_1, N_2, ...N_n\}$. Let $numNets$ denote the number of simultaneous networks to which the card is associated at a particular instant, and T_i denote the activity period, ATP$_i$, i.e. the time a card stays in a network N_i. Given these notations the states in Figure 3 are explained as follows:

- START: Cards start in this state when they are either not connected to a wireless network, or are not using MultiNet to connect to them. They might be connected to at most one wireless network.
- INIT N_j: Wireless cards enter this state when they want to join a new wireless network N_j. They can enter this state either from the 'START' state, or the 'ACTIVE N_i' state described later. On entering the 'INIT N_j' state, the card sets up a virtual adapter for network N_j if it does not exist. After creating the virtual adapter, it synchronizes with other nodes if N_j is an ad hoc network. Nodes also set up the data structures for buffering and maintaining other information in this state.

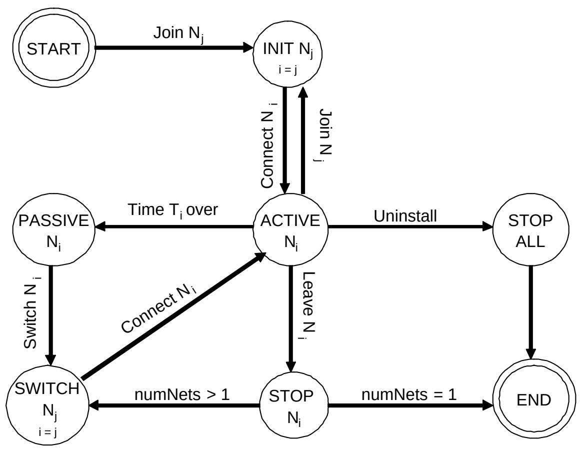


Fig. 3. The states of a MultiNet node

- ACTIVE N_i: The wireless card is connected to wireless network N_i in this state. Packets sent over this network are sent based on the scheme described in subsections V-B and V-D, and the packets sent over other networks $N_k \neq N_i$ are buffered to be sent later.
- PASSIVE N_i: After the wireless card has spent a time T_i in network N_i it moves to the 'PASSIVE N_i' state. The network stack corresponding to N_i is deactivated. Consequently, all packets sent over N_i are buffered till the corresponding stack is activated later in state 'SWITCH N_j'. The state corresponding to N_i is stored, and is used to switch back to this network later on.
- SWITCH N_j: Wireless cards enter this state when the current network N_i is either removed in state 'STOP N_i', or the time T_i in it has expired. In both these cases, cards use the switching strategy to determine the next wireless network, N_j, to connect to, and the time, T_j, the card should stay on it. The card then connects to network N_j, activates the corresponding network stack, and sends all the packets that were buffered on it. Then the card sets $i = j$ and moves to state 'ACTIVE N_i'.
- STOP N_i: Cards have the option of leaving a network. They enter this state if they want to leave a network N_i. All the packets for this network are canceled and the virtual adapter for this network is destroyed. If MultiNet is still used to connect to more than one wireless network, i.e. $numNets > 1$, the card goes to the 'SWITCH N_j' state, and connects to the next network. Otherwise, the card goes to 'END' state.
- STOP ALL: Cards get to this state when MultiNet is to be uninstalled from the network interface. MultiNet cancels the packets buffered for all the $numNets$ networks, and destroys the corresponding virtual adapters. Then the card moves to the 'END' state.
- END: The wireless card is connected to at most one network in this state. MultiNet is uninstalled from the network interface, and traditional techniques are used to connect to the singular wireless network.

VI. Implementation

We have implemented MultiNet on Windows XP and tested it on small scale IEEE 802.11 networks. Windows provides a Network Driver Interface Specification (NDIS) as an intermediate layer between the network device drivers and IP. [1] We implemented MultiNet as a combination of an NDIS intermediate driver and a service (daemon). The service has the buffering and switching logic; it passes instructions to the driver, which implements the mechanics for this buffering and switching. We require all the wireless nodes to implement MultiNet. *However, no changes are required in the wired nodes for MultiNet to work.* The APs too do not require any modification if the IEEE 802.11 PSM is used, as described in Section V-B.4. We note that MultiNet still works if other wireless nodes do not run MultiNet, or the APs do not buffer packets. However, the performance degrades significantly in these cases. In the rest of this section we describe the details of our implementation.

[1]NDIS provides transport independence for the network card vendors since all the upper layer protocols call the NDIS interface to access the network.

A. MultiNet Driver

To the Windows NDIS interface we added an intermediate driver, which we refer to as the MultiNet Driver. NDIS requires the lower edge of the network protocol driver to bind to a network miniport driver [2], and the upper edge of miniport drivers to bind to a protocol driver. Therefore the MultiNet Driver comprises two components: the MultiNet Protocol Driver (MPD) that binds at the lower edge to the network card miniport driver, and the MultiNet Miniport Driver (MMD) that binds at the upper edge to the network protocols, such as TCP/IP. The modified stack is illustrated in Figure 4. The MPD exposes a virtual adapter for each network to which the wireless card is connected. The MMD maintains the state for each virtual adapter. The advantage of this architecture is that there is a different IP address for each network.

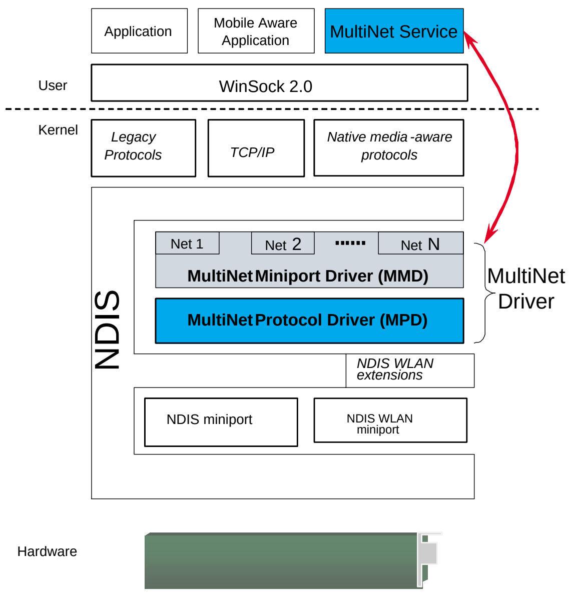


Fig. 4. The modified Windows network stack

The network stack sees each virtual adapter as a different wireless card interface. Ideally, each of these virtual adapters should have a distinct MAC address. Although it is possible to do this over an IEEE 802.3 (Ethernet) interface, commercially available IEEE 802.11 cards do not forward packets from another MAC address. Therefore, each virtual adapter is given the MAC address of the underlying wireless card.

The MPD manages the state of the virtual adapters. It switches the association of the underlying card across different networks, and buffers packets if the SSID of the associated network is different from the SSID of the sending virtual adapter. MPD also buffers packets on the instruction of the MultiNet Service (described later). The MPD also handles packets received by the wireless adapter. A wireless card can send and receive packets only on the currently associated network. A packet received on the wireless adapter is sent to the virtual adapter that is active at that instant. The information about the currently active virtual adapter is maintained by the MPD.

The MMD maintains the state for each virtual adapter. This includes the SSID and operational mode of the wireless network. It is also responsible for handling query and set operations meant for the underlying wireless adapter.

B. MultiNet Service

In addition to the MultiNet Driver, the other major software component is the MultiNet Service. This service is implemented at the user level and implements the buffering and switching logic. It interacts with other MultiNet nodes, and passes signaling messages to the MultiNet Driver to either start or stop a switching and buffering action. The MultiNet Service is responsible for signaling the switching time to the MPD. This signal indicates the time to switch the card, and activate another network. We have implemented the strategies described in Section V-C to determine the Activity Period of a network.

[2]A miniport driver directly manages a network interface card (NIC) and provides an interface to higher-level drivers.

The MultiNet Service on the switching node broadcasts a message just before the MPD switches the card to another network. This message is received by the MultiNet Service running on other MultiNet nodes, and indicates the state of the sending card. The MultiNet Service on the receiving nodes indicates this state to its MultiNet Driver, which then starts to buffer packets.

The MultiNet Service uses I/O Control Codes (ioctls) to interact with the MultiNet Driver. Query and set ioctls are implemented for exposing the features and managing the behavior of the driver. We note that although the switching signal logic can be implemented in the MPD using the NDIS timer, for ease of programming we implement it at the user-level.

C. Buffering Protocol

We discussed in Section V-B.4 that buffering over infrastructure networks can be achieved by using the IEEE 802.11 PSM. We successfully implemented this scheme over NextGen 802.11 cards described in Section III-B. For legacy cards, we were constrained by the proprietary software on them that do not expose any API in Windows to programmatically set the resolution of power save mode. Therefore, we were unable to implement the buffering algorithm for these WLAN cards as described previously. We get around this problem by buffering packets at the end points of the infrastructure networks, using a similar scheme described for ad hoc networks in Section V-B.2. The MultiNet Service keeps track of the end points of all on-going sessions, and buffers packets if the destination is currently in another network. For wide scale deployment, it is unreasonable to expect Internet servers, such as Yahoo or Amazon, to do the buffering for MultiNet nodes, but it is practical to implement the buffering algorithm at the APs as described in Section V-B.4. Fortunately, with NextGen 802.11 based cards and APs, all buffering is possible in the AP's operating system, and implementing our scheme is not a problem.

D. Interaction with Zero Configuration

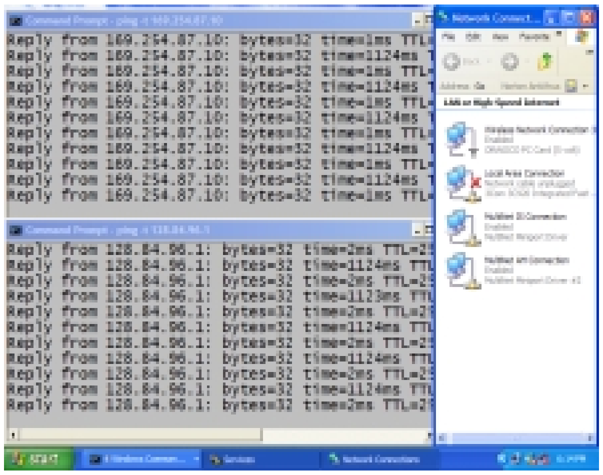
Wireless Zero Configuration (WZC) is a service in Windows XP that forces connectivity to the first available wireless network in a user's list of 'Preferred Networks'. This feature of WZC interferes with the correct functionality of MultiNet, since there is not a single 'Preferred Network' in MultiNet, but multiple wireless networks to which connectivity is desired. Therefore, we have to stop WZC for MultiNet to function correctly. WZC implements the client 802.1X [11] protocol for wireless network authentication, and we are currently modifying the WZC code to not force network connectivity, and run MultiNet along with it, to provide the best features of both WZC and MultiNet. Alternatively, it is possible to treat the different virtual miniports as different wireless adapters to the user, and then allow WZC to have a preferred network for every virtual adapter.

E. Addressing

MultiNet requires network dependent IP addresses for the different virtual adapters exposed by it. However, this turned out to be a difficult constraint. In particular, an ad hoc network works in Windows by assigning the node an autonet address. i.e. 169.254.x.y address. In Windows XP, a node is assigned an autonet address only if it is unable to get a proper DHCP address. This scheme works fine if the network card stays in the ad hoc node for some time. However, in our implementation, the network card periodically switches between networks. If one of the networks is an infrastructure network, or a network with a DHCP server, the DHCP request gets through and the ad hoc node gets an IP address outside the range of autonet addresses. We fixed this problem by manually allocating IP addresses for the virtual miniports. We present a better solution in Section VIII.

F. MultiNet in Operation

Figure 5 shows a screen snapshot of a Windows XP machine connected to two networks, one infrastructure and one ad hoc, using MultiNet. The figure shows two simultaneous ping sessions to machines in different networks. The snapshot machine is able to successfully communicate with the two machines, 169.254.87.10 in the ad hoc network, and 128.84.96.1 in the infrastructure network. Figure 5 also shows the networks visible to the user in the 'Network Connections' window. The first two icons in this window are usually seen by the user, even without MultiNet. The first one, called 'Wireless Network Connection 3' corresponds to the wireless card used by the computer, and the second one labeled 'Local Area Connection' corresponds to the Ethernet card that is inactive at this time. The other icons are generated by MultiNet. In this case the two icons, labeled ‘MultiNet IS Connection’ and ‘MultiNet AH Connection’ correspond to the infrastructure and ad hoc networks that this user is connected to. Each of these icons export a miniport instance to the user, who can see different IP addresses for the two networks. Further, the user sees both the networks as active all the time. Therefore, the view exported to the user is that of different wireless cards with each one corresponding to each network that she is connected to. This feature is consistent with our design described in Section V-A.



```
Reply from 169.254.87.10: bytes=32 time=1ms TTL=
Reply from 169.254.87.10: bytes=32 time=1124ms T
Reply from 169.254.87.10: bytes=32 time=1ms TTL=
Reply from 169.254.87.10: bytes=32 time=1124ms T
Reply from 169.254.87.10: bytes=32 time=1ms TTL=
Reply from 169.254.87.10: bytes=32 time=1124ms T
Reply from 169.254.87.10: bytes=32 time=1ms TTL=
Reply from 169.254.87.10: bytes=32 time=1124ms T
Reply from 169.254.87.10: bytes=32 time=1ms TTL=
Reply from 169.254.87.10: bytes=32 time=1124ms T
Reply from 169.254.87.10: bytes=32 time=1ms TTL=
```

```
Reply from 128.84.96.1: bytes=32 time=2ms TTL=2
Reply from 128.84.96.1: bytes=32 time=1124ms TTL
Reply from 128.84.96.1: bytes=32 time=2ms TTL=2
Reply from 128.84.96.1: bytes=32 time=1123ms TTL
Reply from 128.84.96.1: bytes=32 time=2ms TTL=2
Reply from 128.84.96.1: bytes=32 time=1124ms TTL
Reply from 128.84.96.1: bytes=32 time=2ms TTL=2
Reply from 128.84.96.1: bytes=32 time=1124ms TTL
Reply from 128.84.96.1: bytes=32 time=2ms TTL=2
Reply from 128.84.96.1: bytes=32 time=1124ms TTL
Reply from 128.84.96.1: bytes=32 time=2ms TTL=2
```

Fig. 5. A screen snapshot of a MultiNet node connected to two networks using legacy 802.11 cards

VII. System Evaluation

We studied the performance of MultiNet using a real implementation and a custom simulator. The implementation was used to study the throughput behavior with different switching algorithms. We then simulated MultiNet and compared it with the alternative approach of using multiple radios to connect to multiple networks. We compare the two approaches with respect to energy consumption and the average delay encountered by the packets. The results presented in this section confirm that MultiNet is a more efficient way of achieving connectivity to multiple networks as compared to using multiple radios.

A. Test Configuration

MultiNet was deployed on two laptops running Windows XP Professional: a Compaq Evo N600c and an HP Omnibook 6000. The Compaq laptop had a built in IEEE 802.11b card and we added two more cards; one each in the PCMCIA slots of both the laptops to build a three node network. The two other cards were a Cisco 340 Series and an Orinoco Gold Wireless card. We have also tested MultiNet on the latest NextGen 802.11 cards available to us from AMD and Realtek. All these cards have a maximum data rate of 11 Mbps. We were able to test this implementation for four different IEEE 802.11b APs: a Cisco 340 Series, an EZConnect 2656, a DLink DI-614+ and NextGen 802.11 APs. The results were consistent across these network equipment. Most of our experiments were run in a two network setting, an ad hoc and an infrastructure network.

B. Switching Delay

Good performance of MultiNet depends on a short delay when switching across networks. However, legacy IEEE 802.11b cards perform the entire association procedure every time they switch to a network. We carried out a detailed analysis of the time to associate to an IEEE 802.11 network. We placed a laptop in the vicinity of the two MultiNet machines and installed an IEEE 802.11 wireless LAN packet analyzer called AiroPeek [6] on it. AiroPeek supports higher level network protocols such as TCP/IP and fully decodes IEEE 802.11a and IEEE 802.11b WLAN protocols. It is used for analyzing wireless network performance with accurate identification of signal strength, channel and data rates. The 802.11 messages sent when switching to an ad hoc network are illustrated in Figure 6. Figure 7 shows the steps on switching to an infrastructure network. Of particular interest is the significant overhead when switching to a network.

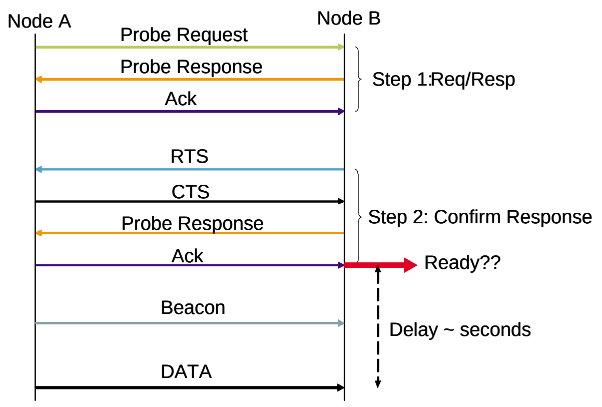


Fig. 6. Message exchange, as seen by AiroPeek, when switching to an ad hoc network on the Cisco 340 Series wireless adapter

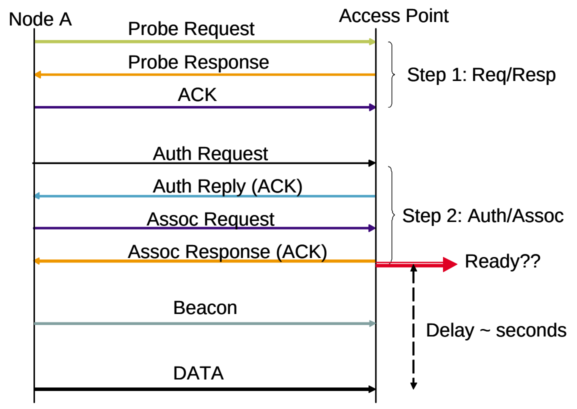


Fig. 7. Message exchange, as seen by AiroPeek, when switching to an infrastructure network on the Cisco 340 Series wireless adapter

Looking at Figures 6 and 7, we see a significant overhead when switching from one network to another. In fact, a more baffling characteristic is the amount of delay. The delay is an astronomical delay of 3.9 seconds was observed from the time the card started associating to an ad hoc network, after switching from an infrastructure network, to the time it started sending data.

TABLE I

THE DELAYS ON SWITCHING BETWEEN IS AND AH NETWORKS FOR IEEE 802.11 CARDS WITH AND WITHOUT THE OPTIMIZATION OF TRAPPING MEDIA CONNECT AND DISCONNECT MESSAGES.

Switching From	Unoptimized Legacy	Optimized Legacy	Optimized NextGen 802.11
IS to AH	3.9 s	170 ms	25 ms
AH to IS	2.8 s	300 ms	30 ms

Our investigations revealed that the cause of this delay is the *media disconnect* and *media connect* notifications to the IP stack. The IP stack damps the media disconnect and connect for a few seconds to protect itself and its clients from spurious signals. The spurious connects and disconnects can be generated by network interface cards due to a variety of reasons ranging from buggy implementations of the card or switch firmware to the card/switch resetting itself to flush out old state. Windows was designed to damp the media disconnect and connect notifications for some time before rechecking the connectivity state of the adapter and taking the action commensurate with that state.

In the case of MultiNet, switching between networks is deliberate and meant to be hidden from higher protocols such as IP and its clients. We hide switching by having MPD trap the media disconnect and media connect messages when it switches between networks. Since the MPD is placed below IP, it can prevent the network layer from receiving these messages. This minor modification results in significant improvement in the switching overhead as shown in Table I. Using the above optimization, we were able to reduce the switching delay from 2.8 seconds to 300 ms when switching from an ad hoc network to an infrastructure network and from 3.9 seconds to 170 ms when switching from an infrastructure network to an ad hoc network. *These numbers are further reduced to as low as 30 ms and 25 ms respectively, when NextGen 802.11 cards are used.* We believe that this overhead is extraneous for purposes of MultiNet and in Section VIII we suggest additional ways to make this delay negligible.

A nice consequence of masking the media connect and media disconnect messages is that all virtual adapters are visible to IP as active, and our architecture of Section V-A is therefore consistent.

C. Switching Strategies

We implemented three switching strategies described in Section V-C, i.e. Fixed Priority, Adaptive Buffer, and Adaptive Traffic. The test environment had a MultiNet node connected to an infrastructure and an ad hoc network. The time to switch to the ad hoc and infrastructure networks were overestimated at 500ms and 300ms respectively. [3] The total time available for switching between networks was 1 sec. We evaluated the switching strategies when simultaneously transferring a file of size 47 MB using FTP from the MultiNet node to two nodes on the different networks. An independent transfer of the file over the ad hoc network took 80.25 seconds, while it took 54.12 seconds over the infrastructure network.

Figure 8 shows the time taken to simultaneously transfer this file over MultiNet using different switching strategies for legacy cards. We evaluated 3 different Fixed Priority switching schemes. In the ‘50%IS 50%AH’ strategy the node stays on each network for 500ms. In the ‘75%IS 25%AH’ scheme it stays on the infrastructure network for 750ms and on the ad hoc network for 250ms, and in the ‘25%IS 75%AH’ scheme the node stays on the infrastructure network for 250 ms and the ad hoc network for 750ms. For the Adaptive Traffic algorithm we used a window of 3 switching cycles to estimate the Activity Periods. In this case the window is 3*1.8 = 5.4 seconds since a switching cycle is 500+300+1000 = 1800 ms.

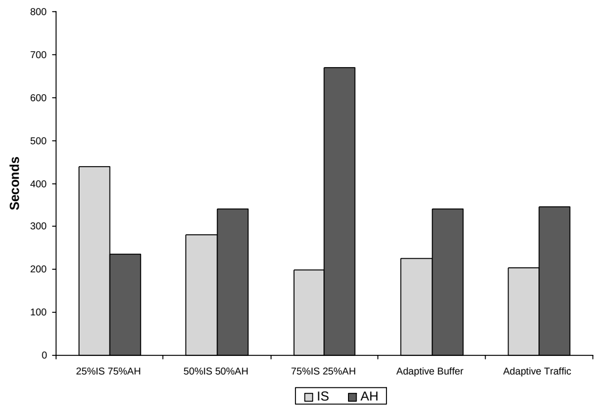


Fig. 8. Amount of time taken to complete the FTP transfer of a file on an ad hoc and infrastructure network for different switching strategies

Different switching strategies show different behavior and each of them might be useful for different scenarios. For the fixed switching strategies the network with higher priority gets a larger slot to remain active. Therefore, the network with a higher priority takes lesser time to complete the FTP transfer. The results of the Adaptive algorithms are similar. The Adaptive Buffer algorithm adjusts the time it stays on a network based on the number of packets buffered for that network. Since the maximum throughput on an infrastructure network is more than the throughput of an ad hoc network [4], the number of packets buffered for the infrastructure network is more. Therefore the FTP transfer completes faster over the infrastructure network as compared to the ‘50%IS 50%AH’ case. For a similar reason the FTP transfer over the infrastructure network completes faster when using

[3]This overprovisioning was necessary to avoid any loss of packets in case the switching failed in this time. With NextGen 802.11 cards these parameters were reduced to 50ms.

[4]Separate experiments revealed that the *average* throughput on a wireless network with commercial APs and wireless cards is 5.8 Mbps for an isolated infrastructure network and 4.4 Mbps for an isolated two node ad hoc network. These results are consistent with [10].

Adaptive Traffic switching. MultiNet sees much more traffic sent over the infrastructure network and proportionally gives more time to it. Overall, the adaptive strategies work by giving more time to faster networks if there is maximum activity over all the networks. However, if some networks are more active than the others, then the active networks get more time. We expect these adaptive strategies to give the best performance if the user has no priority and wants to achieve the best performance over all the MultiNet networks.

D. Adaptive Switching

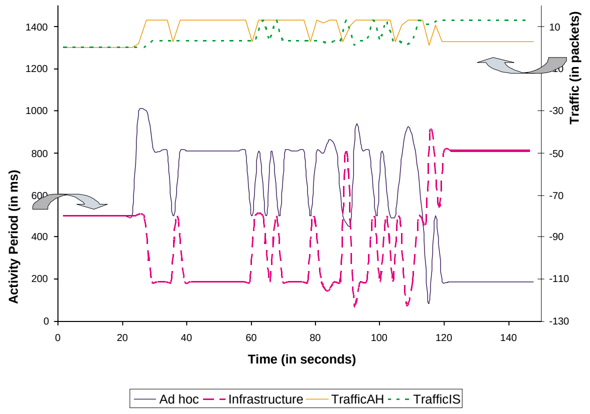


Fig. 9. Variation of the activity period for two networks with time. The activity period of a network is directly proportional to the relative traffic on it.

The adaptability of MultiNet is demonstrated in Figure 9. The Adaptive Buffer switching strategy is evaluated by running our system for two networks, an ad hoc and an infrastructure network, for 150 seconds. The plots at the top of Figure 9 show the traffic seen on both the wireless networks, and the ones at the bottom of this figure show the corresponding effect on the activity period of each network. As a result of the adaptive switching strategy the activity period of the networks vary according to the traffic seen on them. Initially when there is no traffic on either network, MultiNet gives equal time to both networks. After 20 seconds there is more traffic on the ad hoc network, and so MultiNet allocates more time to it. The traffic on the infrastructure network is greater than the traffic on the ad hoc network after around 110 seconds. Consequently, the infrastructure network is allocated more time. This correspondence between relative traffic on a network and its activity periods is evident in Figure 9.

MultiNet, when used with adaptive switching schemes, provides true zero configuration. Prior schemes, such as Wireless Zero Configuration (WZC), require users to specify a list of preferred networks, and WZC only connects to the most preferred available wireless network. The adaptive switching strategies require a user to specify a list of preferred networks, and the card connects to all the networks giving time to a network based on the amount of traffic on it.

E. Mixed Deployments with Legacy Cards

We have implemented buffering on infrastructure networks with NextGen 802.11 cards using the IEEE 802.11 PSM. MultiNet can also be used with legacy cards that do not allow us to set the PSM resolution using the scheme presented in Section VI-C. Remote nodes might not buffer packets and we show the performance degradation for a two network MultiNet in Figure 10. Packets were sent, using ntttcp, over the infrastructure network from the MultiNet node to another node in the network. Ntttcp, which is a port of ttcp [17] to Windows, works by establishing a TCP session between two nodes and sending the packets at the maximum rate. The activity period for both the networks was fixed at 500 ms. We present results for three scenarios in Figure 10. 'NoMultiNet' corresponds to the case when the sender and receiver are connected to just one network, 'MultiNetNoBuffer' is when the sender is connected to two networks using MultiNet while the receiver does not buffer packets, and 'MultiNetBuffer' is the scenario when the receiver buffers packets for the MultiNet node. The results show that the performance drops by a factor of four when using MultiNet with buffering and drops further when the receiver does not buffer packets. Without buffering the throughput of the system goes down to a seventh of the maximum achievable throughput. Although performance drops significantly, MultiNet is still usable with a throughput of around 500 Kbps.

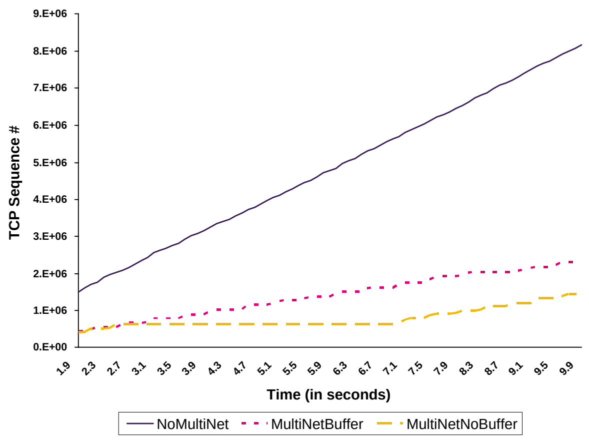


Fig. 10. TCP Performance in mixed deployment settings: with and without MultiNet buffering.

F. MultiNet versus Multiple Radios

MultiNet is one way of staying connected to multiple wireless networks. The alternative approach is to use multiple wireless cards. Each card connects to a different network, and the machine is therefore connected to multiple networks. We simulated this approach, and compared it with the MultiNet scheme with respect to the energy consumed and the average delay of packets over the different networks. We first present our simulation environment, and then compare the results of the MultiNet scheme to the alternative approach using multiple radios.

1) Simulation Environment: We simulated both approaches for a sample scenario of people wanting to share and discuss a presentation over an ad hoc network and browse the web over the infrastructure network at the same time. We model traffic over the two networks, and analyze the packet trace using our simulator.

Traffic over the infrastructure network is considered to be mostly web browsing. We used Surge [7] to model `http` requests according to the behavior of an Internet user. Surge is a tool that generates web requests with statistical properties similar to measured Internet data. The generated sequence of URL requests exhibit representative distributions for requested document size, temporal locality, spatial locality, user off times, document popularity and embedded document count. For our purposes, Surge was used to generate a web trace for a 1 hour 50 minute duration, and this web trace was then broken down to a sample packet trace for this period. The distribution of the packet sizes over the infrastructure network is illustrated in Figure11.

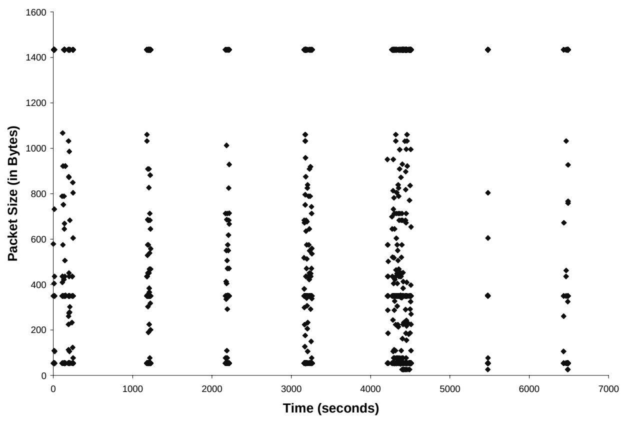


Fig. 11. Packet trace for the web browsing application over the infrastructure network

The ad hoc network is used for two purposes: sharing a presentation, and supporting discussions using a sample chat application. Three presentations are shared in our application over a 1 hour 50 minute period. Each presentation is a 2 MB file, and is downloaded to the target machine using an FTP session over the ad hoc network. They are downloaded in the 1st minute, the 38th minute, and the 75th minute. Further, the user also chats continuously with other people in the presentation room, discussing the presentation and other relevant topics. Packet traces for both the applications, FTP and chat, were obtained by sniffing the network, using Ethereal [3], while running the respective applications. MSN messenger was used for a sample chat trace for a 30 minute duration. The Packet traces for FTP and chat were then extended over the duration of our application, and are illustrated in Figure 12.

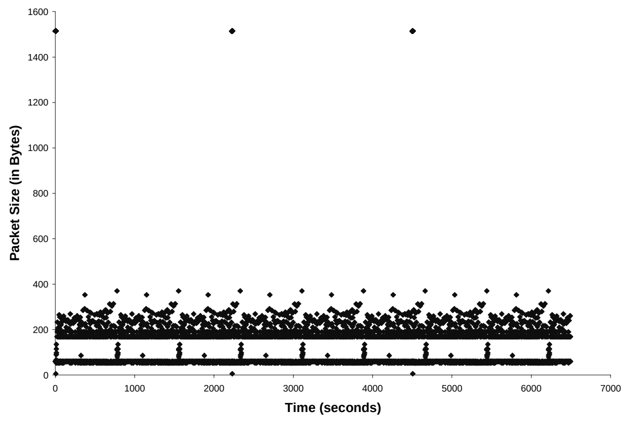


Fig. 12. Packet trace for the presentation and chat workloads over the ad hoc network

In our simulations we assume that wireless networks operate at their maximum TCP throughput of 4.4 and 5.8 Mbps for an ad hoc and infrastructure network respectively. We then analyze the packet traces for independent networks, and generate another trace for MultiNet. We use a '75%IS 25%AH' switching strategy presented in Section VII-C with a switching cycle time of 400ms. The switching delay is set to 1 ms, and we explain the reason for choosing this value in Section VIII-A. Further, the power consumed when switching is assumed to be negligible[5]. We do not expect these simplifying assumptions to greatly affect the results of our experiments. We analyze packet traces for the two radio and MultiNet case and compute the total power consumed and the average delay encountered by the packets. All the cards are assumed to be Cisco AIR-PCM350, and their corresponding power consumption numbers are used from [16]. Specifically, the card consumes 45 mW of power in sleep mode, 1.08W in idle mode, 1.3W in receive mode, and 1.875W in transmit mode. Further, in PSM, the energy consumed by the Cisco AIR-PCM 350 in one power save cycle is given by: $0.045 * n * t + 24200$ milliJoules, where n is the Listen Interval and t is the Beacon Period of the AP. The details of these numbers are presented in [16].

2) Without Power Save Mode: We analyze the two schemes of connecting to multiple networks with respect to the performance on the network and the amount of power consumed. In our simulated scenario, each of the radios gives the best achievable throughput on both the networks. As shown in Table II, the average throughput of MultiNet in the infrastructure mode is 4.35 Mbps compared to 5.8 Mbps in the two radio case. The average throughput in the ad hoc network is 1.1 Mbps in MultiNet and 4.4 Mbps when using two radios. Switching results in lesser throughput across individual networks, since it is on a network for a smaller time period. Consequently, the scheme of using multiple cards gives much better throughput as compared to MultiNet when connected to multiple networks.

TABLE II

THE AVERAGE THROUGHPUT IN THE AD HOC AND INFRASTRUCTURE NETWORKS USING BOTH STRATEGIES OF MULTINET AND TWO RADIOS

Network	Two Radio	MultiNet
Ad Hoc	4.4 Mbps	1.1 Mbps
Infrastructure	5.8 Mbps	4.35 Mbps

However, the scheme of using multiple radios consumes more power. Each radio is always on, and therefore keeps transmitting and receiving over all the networks. Even when it is not, the radio is in idle mode, and drains a significant amount of power.

[5]Note switching between networks is different from switching between various power levels.

Figure 13 shows the amount of energy consumed by the MultiNet scheme and the two radio scheme for the above application. Two radios consume around double the power consumed by the single MultiNet radio.

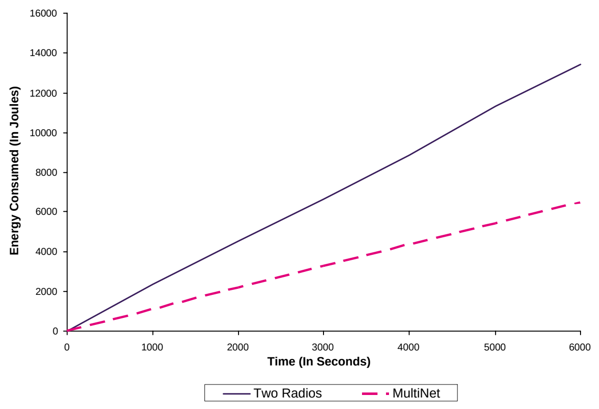


Fig. 13. Comparison of total energy usage when using MultiNet versus two radios

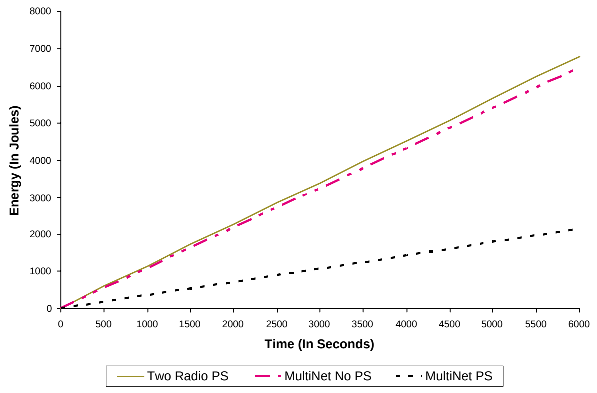


Fig. 14. Energy usage when using MultiNet and two radios with IEEE 802.11 Power Saving

3) With Power Save Mode: The multiple radio approach can be modified to consume less power by allowing the network card in infrastructure mode to use PSM. Figure 14 shows the energy usage when the infrastructure radio uses PSM for our application. The Beacon Period is set to 100 ms, and the Listen Interval is 4. The amount of energy consumed in the two radio case using PSM is very close to the consumption of MultiNet without PSM. However, this saving comes at a price. It is no longer possible to achieve the high throughput for infrastructure networks if the cards are in PSM. Simulated results in Table III show that the average packet delay over the infrastructure network with PSM is now close to the average packet delay for MultiNet. *Therefore, using two radios with PSM does not give significant benefits as compared to MultiNet without PSM.*

TABLE III

THE AVERAGE PACKET DELAY IN INFRASTRUCTURE MODE FOR THE VARIOUS STRATEGIES

Scheme	Avg Delay (in Seconds)
Two Radio	0.001
MultiNet	0.157
Two Radio PS	0.156
MultiNet PS	0.167

The power consumption of MultiNet can be reduced further by allowing it to enter the power save mode for infrastructure networks as described in Section V-C.2. In our experiment we chose the Switching Cycle to be 400ms, with ‘75%IS 25%AH’ switching. For consistency in comparison, the Listen Interval is set to 4 and the Beacon Period to 100ms. Consequently, every time the card switches to infrastructure mode, it listens for the traffic indication map from the AP. After it has processed all its packets it goes to sleep and wakes up after 300ms. It then stays in the ad hoc network for 100ms, and then switches back to the infrastructure network. The modified algorithm results in greater energy savings as shown in Figure 14. The average delay per packet over the infrastructure network is not seriously affected, while the energy consumed is reduced by more than a factor of 3. We conclude that MultiNet beats using multiple cards for connecting to multiple networks in terms of convenience and power.

Note that we do not evaluate power saving in ad hoc mode because we are unaware of any commercial cards that implement this feature. As a result we were unable to get performance numbers when using PSM in ad hoc mode. However, we believe that if such a scheme is implemented, we will be able to incorporate it in MultiNet, and further reduce the power consumption.

G. *Maximum Connectivity in MultiNet*

What is the maximum number of networks that can be supported by MultiNet? The answer to this question depends on the performance features desired by the user on each network. Essentially, it is a tradeoff between connectivity and latency, and the onus is on users to decide whether they are willing to forego a drop in throughput for the extra connectivity. We use the simulation environment of Section VII-F.1 to evaluate the performance on increasing the number of connected networks. Table IV presents the average delay seen by packets over the infrastructure network on varying the number of MultiNet networks from 2 to 6. We used a Fixed Priority switching strategy with equal priorities to all the networks. An increase in the number of connected networks results in a smaller Activity Period for each connected network when using Fixed Priority Switching. As a result, more packets are buffered and the average delay encountered by the packets on a network increases. This is shown in Table IV.

TABLE IV

THE AVERAGE PACKET DELAY IN INFRASTRUCTURE MODE ON VARYING THE NUMBER OF MULTINET CONNECTED NETWORKS

Num Networks	Avg Delay (in Seconds)
2	0.191
3	0.261
4	0.332
5	0.410
6	0.485

H. *Summary*

We summarize the conclusions of our performance analysis as follows:

- No single switching strategy is best under all circumstances. Adaptive strategies are best when no network preference is indicated. Both Adaptive Buffer and Adaptive Traffic give similar performance
- For the applications studied, MultiNet consumes 50% less energy than a two card solution.
- As expected, the average packet delay with MultiNet varies linearly with an increase in the number of connected networks when all the networks are given equal activity periods.
- MultiNet works even when the access point does not buffer packets for the switching node, although the performance goes down by a factor of 4.
- Masking ‘media connects’ and ‘media disconnects’ below IP leads to significant reduction in the switching overhead. The switching delay for legacy cards is reduced to around 300 ms, while this number goes down to 30 ms for NextGen 802.11 cards.
- Adaptive Switching eliminates the current zero configuration requirement to prioritize the preferred network. With MultiNet based zero configuration, the user connects to all preferred networks.

VIII. Discussion

We now discuss various ways in which the performance of MultiNet can be improved further. In particular, we focus on reducing the switching overhead, enabling 802.1X [11] authentication, and deployment.

A. Reducing the Switching Overhead

Good performance of MultiNet depends on low switching delays. The main cause of the switching overhead in current generation wireless cards is the 802.11 protocol, which is executed every time the card switches to a network. The card believes that it has disassociated from the previous network, and starts association afresh. Further, these cards do not store state for more than one network in the firmware, and worse still, many card vendors force a firmware reset when changing the mode from ad hoc to infrastructure and vice versa.

Most of these problems are fixed in the NextGen 802.11 cards. These cards do not incur a firmware reset on changing their mode. Moreover, since switching is forced by MultiNet, NextGen 802.11 cards do not explicitly disconnect from the network when switching. However, they still carry out the association procedure that causes the 25 to 30 ms delay. By allowing upper layer software to control associations, instead of automatically initializing them, this delay can be made negligible. The only overhead on switching is then the synchronization with the wireless network. This can be done reactively, with the card requesting a synchronization beacon when it switches to a network.

Using the above optimizations, a WLAN card can switch to a network as fast as the network's state can be loaded into a flash card. Since the network state to load is around 100 bytes, and data transfer speeds for flash cards is 8Mbps [2], **we expect the switching overhead to be less than 1 ms.**

B. Network Port Based Authentication

For MultiNet to be useful in all environments it has to support this authentication protocol. However, the supplicant 802.1X protocol is implemented in the Wireless Zero Configuration Service (WZC) for Windows XP, and we had to turn off WZC for MultiNet to work. Only minor changes are needed in WZC for it to work with MultiNet. However, achieving good performance with IEEE 802.1X is difficult. We measured the overhead of the IEEE 802.1X authentication protocol and found it to be approximately 600 ms. It is clear that we need to prevent the card from going through a complete authentication procedure every time it switches across IEEE 802.1X enabled networks. We can eliminate the authentication cycles by storing the IEEE 802.1X state in the MPD and using this state instead of redoing the authentication procedure. Further, the IEEE 802.11 standard recommends an optimization called 'Preauthentication' for the APs. Preauthentication works by having the APs maintain a list of authenticated nodes. When implemented, this optimization will eliminate the authentication overhead every time the wireless card switches to an 802.1X enabled network.

C. Route Tables

Although MultiNet has support for multiple networks, we do not discuss the details of modifying the route table here. If the destination is not on any of the connected networks, MultiNet chooses a default gateway to send the packets. This is done even if the network with the gateway node is currently inactive. To send packets only on gateways of currently active networks, we propose using a flag when deciding the default gateway. This flag should be managed by the MultiNet Service based on the currently active network. This ensures that packets routable on the currently active network are not buffered.

D. Deployability

An important goal in our design of MultiNet was the ease of deployment. Nodes can enable MultiNet service without any major change to the existing software. For MultiNet to work in an infrastructure network, APs need to have IEEE Power Saving Mode [12] enabled. In ad hoc networks, for optimal performance MultiNet requires other nodes in the network to have the ability buffer packets. However, this is not a requirement for MultiNet to work.

E. Can MultiNet be Implemented in the Firmware?

The simple answer is yes, however we strongly advocate that the right place to implement MultiNet is as a kernel driver. Buffering imposes memory requirements that are best taken care of by the operating system, and the policy driven behavior can bloat the firmware. Additionally, by moving the intelligence into a general purpose PC, the cost of the wireless hardware can be reduced further, which is the trend for the next generation of WLAN cards we described in Section III-B

F. Impact of Virtualization

MultiNet enables a framework for the design of a new class of applications. System designers are no longer constrained by the number of wireless cards they can fit into in a system. They are free to design systems and applications that can connect to many wireless networks at the same time. To the best of our knowledge, MultiNet is the first system that relaxes this physical constraint.

IX. Future Research

The switching behavior of MultiNet augurs badly for TCP performance. MultiNet is implemented below IP, and so TCP sees fluctuating behavior for packets sent by it. It receives immediate acknowledgements for packets sent when the network is active, and delayed acknowledgements for buffered packets. The above behavior affects the way TCP adjusts the RTT for the session, and from the way it is calculated, the RTT will always be an upper bound. An overestimate of RTT results in larger timeout values to ensure that packets are not lost. However, a larger than required RTT has other consequences with respect to flow control, and congestion response. This problem is generally relevant for networks that have periodic connectivity. A solution to this problem has to mask the delay encountered by the buffered packets. We are currently exploring ways to achieve this, and improve TCP performance.

As stated previously, we do not consider scenarios when a MultiNet node is participating in a multihop ad hoc network. The synchronization problem is complicated for such scenarios. A scheme that supports multihop networks has to handle partitioning issues of the ad hoc network, and ways to resynchronize it. We are exploring different randomization schemes to ensure probabilistically good synchronization among nodes.

X. Conclusion

We conclude by summarizing the main contributions of our paper as follows:

- We described a new virtualization architecture, called MultiNet, which allows a user to connect to multiple wireless networks by virtualizing the WLAN card. Several compelling real-life scenarios are described that motivate the need for MultiNet. To the best of our knowledge, our paper is the first to articulate this problem and propose a solution for IEEE 802.11 hardware.
- We analyzed two different switching strategies. Non-adaptive fixed switching strategy is better when the different networks are prioritized. Adaptive strategies work by allocating more time to wireless networks with high traffic. This approach has a nice side effect for zero configuration that allows a user to connect to all preferred networks.
- We described a buffering protocol that ensures delivery of packets over switching nodes, and can be implemented at IEEE 802.11 APs using Power Saving Mode.
- We presented a synchronization protocol that ensures successful communication between two switching nodes in an ad hoc network. We implement this scheme over existing hardware and show its performance.

As of this writing our implementation of MultiNet on Windows XP has been operational for over twelve months. During this time period, we have refined the implementation and analyzed our system. We have also identified additional interesting problems, such as the affect of switching on TCP performance and scaling in multihop networks. These problems need to be explored in greater detail and we are working on them actively. We hope to report progress on these and make MultiNet available to the research community in the near future on the following website:
http://research.microsoft.com/∼bahl/MS_Projects/MultiNet/.

References

[1] Advanced Micro Devices (AMD). http://www.amd.com/.
[2] ATA Flash Memory Cards. http://www.magicram.com/flshcrd.htm.
[3] The ethereal network analyzer. http://www.ethereal.com/.
[4] Relatek. http://www.realtek.com.tw/.
[5] VMware: Enterprise-Class Virtualization Software. http://www.vmware.com/.
[6] WildPackets Airopeek. http://www.wildpackets.com/products/airopeek.
[7] Paul Barford and Mark Crovella. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *ACM SIGMETRICS 1998*, pages 151–160, July 1998.

[8] Josh Broch, David A. Maltz, and David B. Johnson. Supporting Hierarchy and Heterogeneous Interfaces in Multi-Hop Wireless Ad Hoc Networks. In *Workshop on Mobile Computing held in conjunction with the International Symposium on Parallel Architectures*, June 1999.

[9] E. Bugnion, S. Devine, and M. Rosenblum. Disco: Running Commodity Operating Systems on Scalable Multiprocessors. In *Sixteenth ACM Symposium on Operating System Principles*, October 1997.

[10] Martin Heusse, Franck Rousseau, Gilles Berger-Sabbatel, and Andrzej Duda. Performance Anomaly of 802.11b. In *IEEE INFOCOM*, 2003.

[11] IEEE. IEEE 802.1x-2001 IEEE Standards for Local and Metropolitan Area Networks: Port-Based Network Access Control. 1999.

[12] IEEE802.11b/D3.0. Wireless LAN Medium Access Control(MAC) and Physical (PHY) Layer Specification: High Speed Physical Layer Extensions in the 2.4 GHz Band. 1999.

[13] Ching Law, Amar K. Mehta, and Kai-Yeung Siu. A New Bluetooth Scatternet Formation Protocol. In *To appear in ACM Mobile Networks and Applications Journal*, 2002.

[14] Ching Law and Kai-Yeung Siu. A Bluetooth Scatternet Formation Algorithm. In *IEEE Symposium on Ad Hoc Wireless Networks 2001*, November 2001.

[15] A. Nasipuri and S. R. Das. Multichannel CSMA with Signal Power-Based Channel Selection for Multihop Wireless Networks. In *IEEE Vehicular Technology Conference (VTC)*, September 2000.

[16] Eugene Shih, Paramvir Bahl, and Mike Sinclair. Wake On Wireless: An Event Driven Energy Saving Strategy for Battery Operated Devices. In *ACM MobiCom 2002*, September 2002.

[17] R. Stine. FYI on a Network Management Tool: Catalog Tools for Monitoring and Debugging TCP/IP Internets and Interconnected Devices. In *IETF RFC 1147*, April 1990.

[18] Andrew Whitaker, Marianne Shaw, and Steven D. Gribble. Scale and Performance in the Denali Isolation Kernel. In *Fifth Symposium on Operating Systems Design and Implementation*, December 2002.