AN INTERLAY ARCHITECTURE FOR WIRELESS MESH NETWORKS

Draft of November 29, 2006 at 21:17

BY

MATTHEW BELCHER

B.S. Computer Science and Mathematics, University of Florida, 2001

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2006

Urbana, Illinois

Abstract

The 802.11[1] PHY/MAC is designed for single hop wireless LANs and the attempts to extend this protocol to multihop wireless meshes have not proved as successful. Notable problems include extremely unstable route thrashing, poor TCP performance, and unfairness in channel sharing due to wireless interference across multiple hops. Nevertheless, there is a strong need to leverage the prevalence of cheap 802.11 devices to build large-scale community wireless mesh. We propose WINE, a wireless interlay network architecture, for this purpose. WINE introduces into the standard Internet TCP/IP protocol stack a new interlay layer, which sits atop 802.11 MAC but below the network layer. WINE interlay layers are only activated in a carefully selected set of mesh nodes (WINE nodes), triggered by transient traffic. Along each traffic route, WINE nodes coordinate among themselves across multiple hops and rate limit transient traffic for interference mitigation. Through intensive simulations we show that WINE improves unmodified TCP throughput by 15%-50%, decreases packet loss by up to 95%, and decreases jitter by more than 90%. WINE also improves the transport of real-time video by approximately 100% in throughput. WINE also significantly improves the stability of routing in a wireless mesh network. In fact, it eliminates almost all route flapping in our simulated scenarios.

[1]We use "802.11" to denote the IEEE 802.11 family.

To my wife and son

Acknowledgments

Thanks to my advisor, Haiyun Luo, for guiding me through this research and for providing hardware, office space, and problem-solving. Further thanks to Michael Earnhart, who assisted in the deployment of our wireless testbed and excellently served as chief BASH scripter. Chandrakanth Chereddi gave us a version of the madwifi driver which he hacked to solve a BSSID problem that plagued us for a week. Finally, thanks to the secretaries of the computer science department, who had patience with me when I left networking hardware in all the hallways.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem

The demand for high-performance, ubiquitous Internet access is driving the development of the next-generation Internet towards wireless. In response, the impact of wireless communications is expanding up the Internet hierarchy from first-/last-hop edges towards the core. Wireless mesh networks for last-mile Internet access have emerged from this trend. The declining cost and increasing speed of 802.11 devices serves as one key factor to realize the vision of wireless mesh networks. However, despite great interest in this new networking paradigm as well as a number of experimental system deployments [4, 2, 3, 10, 28], the 802.11 transmission protocol designed for wireless LANs , i.e. 802.11 Distributed Coordination Function (DCF), does not scale to large multihop wireless mesh [37, 18, 9, 29]. Reports of serious packet drops, unfair channel bandwidth sharing, and extremely unstable routing have been frequently cited to question the feasibility of the 802.11 community wireless mesh. Specifically, extensive evaluations in [34] have shown that the "horizon" of 802.11 wireless networks is as narrow as two to three hops, beyond which the network is no longer useful for TCP flows. Additionally, multi-hop wireless packet loss is dominated by contention and channel errors, not congestion which TCP was designed to reduce. Although it is conceivable to have completely new network architectures that are different from the existing store-and-forward network model [27] and/or PHY/MAC layer (e.g., TAP [22] and directional antenna [26, 33, 11]) specifically designed for large-scale wireless mesh, their high expenses offset wireless mesh's cost advantages over other wired alternatives.

1.2 Background

A wireless mesh network is a network architecture for providing reliable, high-quality Internet service to a wide region using low-cost devices with short broadcast ranges. Each device in the network communicates only with its neighbors in broadcast range. By routing traffic between neighboring nodes, a node on the wireless mesh provides a communication link for nodes which would otherwise have been outside communication range. Since each broadcast range is short, we achieve a greater degree of spatial reuse, since multiple nodes can transmit simultaneously in a wireless mesh network. This has the potential for higher bandwidth than in other wireless architectures such as the cell model, which has a single, high-powered, long-range broadcast.

The 802.11 protocol, designed for wireless LANs, is a popular choice for wireless mesh networks not because it is particularly suited to the task, but rather because 802.11-compatible devices are cheap and ubiquitous. Thus, there is a strong incentive to maintain compatibility with 802.11, despite its probable insuitability for such a task.

Wireless mesh networks introduce new challenges, however. Routing is a complex problem, since a node may be many hops away from the nearest Internet gateway. Other problems include node deployment, out-of-the-network interference, multi-homing, and traffic fairness.

The fundamental challenge in building a large-scale, high-performance 802.11 wireless mesh lies in the fact that wireless transmissions interfere with each other in a range that is usually much larger than the transmission/receiving range (or "one hop" connection range) and dynamic at fine time granularity (one packet transmission time). As a layer-two module 802.11 does not have the explicit information regarding those mesh routers that compete for the shared wireless channel from beyond the communication range. From the perspective of an individual mesh router, the medium access controls are "nearsighted" and do not provide a wired-equivalent stable "link" abstraction on which upper layer modules, e.g., layer-three routing and layer-four transport, can be grounded. Consequently, the routing and transport

protocols, designed for the wired, first-/last-hop wireless Internet, or MANETs, become extremely unstable and unpredictable in a wireless mesh network as the network size grows, especially when the offered load is high. This limit is rooted in the literal inheritance of the layered TCP/IP protocol stack designed for the wired Internet, where the MAC layer is dedicated to one-hop connectivity and it is up to upper-layer routing and transport to deliver packets end-to-end.

Many mechanisms have been proposed to mitigate the problem and help 802.11 scale better. On the one hand, 802.11 DCF variations, packet scheduling [23, 7], and distributed queue management [36] have been proposed and evaluated through simulations. However, they do not address the interferences well. Positioned in the MAC layer they are all restricted by the same limit as 802.11 is. With only one-hop information it is extremely difficult and complex to coordinate access to the shared channel among the mesh routers that cannot directly communicate with each other. On the other hand, end-to-end solutions with global-interference-aware routing [21], and rate-controlled [31, 18], estimation-based [24] transport are proposed. However, since contention for the shared wireless channel evolves at packet-level time granularity, much smaller than the one-way end-to-end delay, end-to-end approaches are often rendered ineffective and impractical.

The 802.11s working group [1] is designing a set of extensions to 802.11 to improve wireless mesh performance.

1.3 Contribution

The above analysis exposes the fundamental limit of the layered TCP/IP Internet architecture when applied in a wireless mesh. Designed for wired networks, none of the layers in the TCP/IP protocol stack is well-positioned to handle the highly dynamic wireless interferences beyond one-hop communication range. In other words, Internet TCP/IP protocol stack does not completely fit into the context of wireless mesh where the concept of "link" is

not well-defined due to long-range wireless interferences. More patches for layer two 802.11 or upper layers do not overcome the architectural limit and, therefore, are unlikely to achieve significant improvement on a large scale. Furthermore, patches usually jeopardize the compatibility with existing 802.11 and Internet protocols, while compatibility with 802.11 is desired to enable immediate wireless mesh deployment and compatibility with legacy upper layer Internet protocols (e.g., routing and TCP) is desired for seamless Internet connectivity - the killer application that motivates the wireless mesh.

In this paper, we tackle the problem through a re-evaluation of the TCP/IP Internet architecture. We choose to introduce a new layer between layer two 802.11 and layer three IP, while keeping the incumbents at upper and lower layers intact for maximum compatibility and immediate deployability. The service provided by this new layer is to help the lower-layer 802.11 better coordinate the access to the shared channel, and provides a stable, wired-equivalent "link" abstract to upper-layer protocols. For transparency to both upper and lower layers, this new layer is implemented as an interlay of entities hosted by a carefully selected set of mesh routers (henceforth called "interlay nodes" or "WINE nodes"). The interlay nodes are deployed multiple hops away from each other at a distance that is close to the wireless interference range. They explicitly coordinate with each other through probing and measurement of the path capacity. Based on the measurement interlay nodes control the offered traffic load to the underlying 802.11 to control the wireless interferences and provide stable, wired-equivalent connections between interlay nodes. We call the new network architecture *WINE: Wireless Interlay Networks*.

Compared with layer two mechanisms, WINE interlay overcomes their nearsightedness through collaboration among WINE nodes that are deployed multiple hops away from each other to handle long-range wireless interferences. Compared with upper-layer end-to-end mechanisms, WINE nodes are close enough for fast response to wireless network dynamics. Finally, a WINE interlay is transparent in the protocol stack. The WINE architecture remains compatible with the 802.11 PHY/MAC and guarantees immediate deployability.

It also remains compatible with upper layer legacy Internet protocols such as TCP and guarantees smooth integration with the wired Internet.

In this paper, we demonstrate the effectiveness of WINE in building high-performance, large-scale 802.11 wireless mesh in terms of packet loss, delay, and jitter. Using extensive simulations we show that WINE reduces the packet loss ratio from 10% to 1%, a value at which existing TCP operates comfortably [12]. It also reduces the delay and jitter by 90%, critical for real-time voice/video transport. For real-time video traces, WINE decreases packet loss by up to 75% and increases throughput by up to 100%. Finally, WINE allows real-time flows to coexist with bulk data transfers. It decreases the packet loss ratio of SSH flows from 50% to less than 1%, when the SSH flows are competing with FTP. We also study the performance of WINE when tied to a dynamic routing protocol. WINE assists routing protocols in keeping stable routes by reducing the number of spurious route errors and rerouting.

1.4 Outline

The rest of this paper is organized as follows. Chapter 1.5 gives an overview of related work. In Chapters 2 and 3 we present the details of WINE network architecture design and interlay node placement strategy. We describe our method for measuring interference in chapter **??**. We provide performance evaluation in Chapter 5.1 on unmodified TCP performance with and without WINE and extend to the evaluation of real-time video traffic performance in Chapter 5.2. We also test in simulation the performance benefits of WINE for the DSR routing protocol in Chapter 5.3. Next, we discuss our implementation of WINE on our wireless testbed and associated performance results. Finally, we conclude in Chapter 7.

1.5 Related Work

Next, we briefly review and compare our work with some related work. Our WINE architecture is related to the design of layer-2.5 that was first coined in IEEE 802.21 to enable handover and interoperability between heterogenous network types. WINE is not designed to handle heterogeneity. In fact, our current WINE design operates on a homogenous 802.11 wireless mesh. WINE is also related to the class of "thin-layer" architectural designs. However, these thin-layer designs usually have to be deployed and enabled in every network node to be effective. In contrast, WINE interlay will *not* be effective if it is enabled on all mesh routers.

WINE architecture bears similarities with overlay [6, 20] and underlay [25] networks. However, both overlay and underlay networks rely on the underlying IP routing functionality to provide advanced services. In contrast, WINE interlay is situated below IP. It coordinates with underlying wireless MAC layer to serve the upper layer protocols.

Recent wireless mesh routing protocol designs [13, 16, 15] do consider interferences and employ on-line monitoring of each "link" through periodic and triggered probing. Although probing may help the system adapt to wireless channel errors caused by environmental noise, independent probing on each "link" cannot accurately characterize interferences. In fact, the interference is partially determined by the traffic distribution as a result of those "link-aware" routing decisions. This inter-dependency in the design loop may seriously hurt the stability of the routing protocols, rendering the performance largely unpredictable. It is therefore not surprising that so far *no convincing evidence has demonstrated high routing or transport performance in wireless mesh networks with more than two to three hops in scale*, either in packet-level simulations or real testbed experiments.

There is a rich literature on mobile ad-hoc network clustering algorithms [8] and core-based routing with minimum connected dominating set [32]. Although they seem to take similar approaches of building virtual infrastructures for efficient routing, those virtual in-

frastructures are all based on the network connectivity and are therefore oblivious to interference. Routing algorithms running on top of an connectivity-oriented, interference-oblivious core will suffer the same way as classic MANET routing protocols and recent "link-aware" mesh routing protocols [13, 15] do.

Chapter 2

Design

The core concept of this paper is a new architecture for multihop wireless meshes. An interlay network is a set of coordinating nodes in a network running a protocol below the network layer but above the MAC layer. One can also think of an interlay protocol as an elaborate interface queue that coordinates with other nodes in the network. We deploy an interlay network through which participating nodes collude to manage the rate of packets within mutual interference regions. Keep in mind that not all the nodes in the wireless network participate in the interlay network. Rather, nodes are selected as interlay nodes based on the interference characteristics of the network so that they lie just outside of the interference range of one another. Nodes which are not selected as interlay nodes follow the default behavior for an 802.11 wireless mesh node. We describe the method for choosing WINE interlay nodes in section 3. This section describes the queueing algorithm and method for rate estimation at each interlay node.

2.1 Architecture overview

We consider an 802.11 wireless mesh network deployed in a two-dimensional space. Every mesh switch is equipped with an 802.11 transceiver operating on a network-wide synchronized channel. Note that multiple channels involve extra latency for channel synchronization and requires time-synchronization hardware, or introduces significant modification on 802.11. Furthermore, multiple unlicensed 802.11 channels are not always available due to spectrum policies at different countries (e.g., Japan). We further assume that network paths are

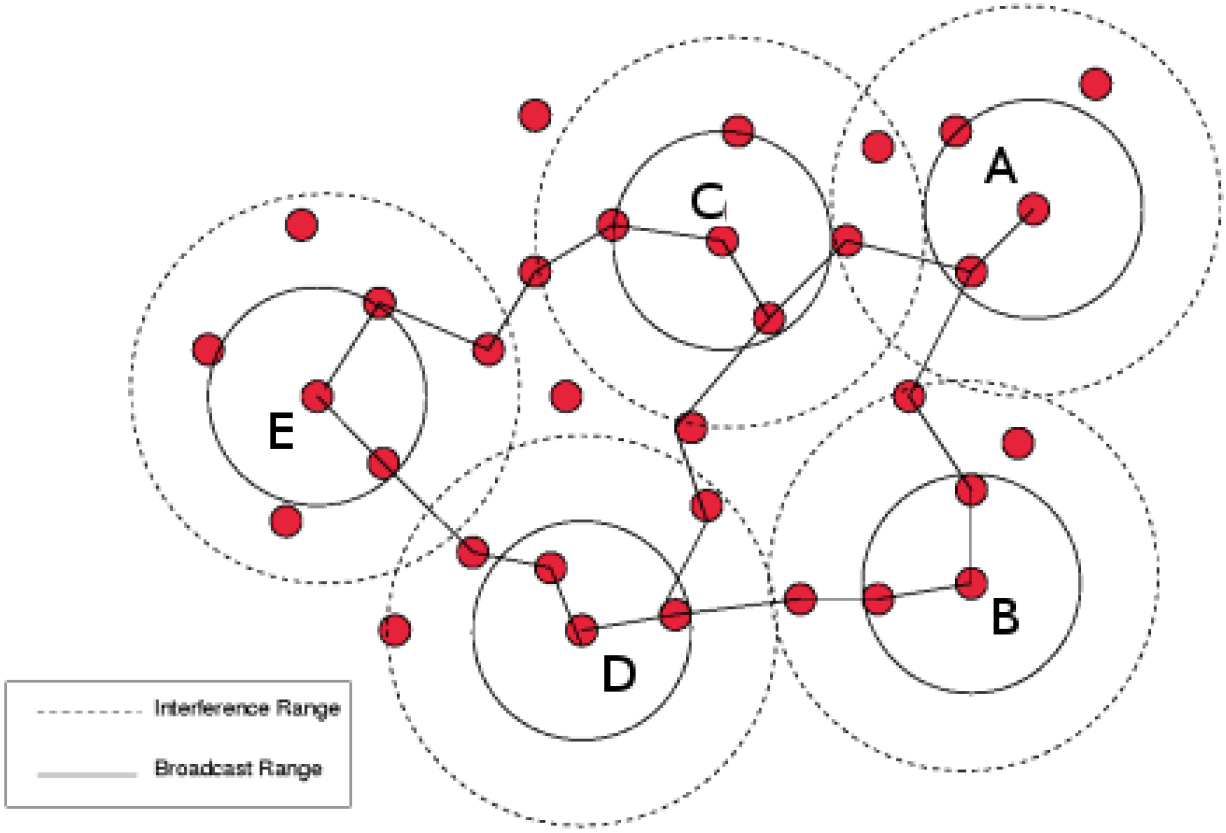

Figure 2.1: An example of a WINE network topology. The communication and interference ranges of interlay nodes are shown.

roughly symmetrical in terms of available bandwidth and delay.

WINE controls the interface queue in the network protocol stack, and therefore it must implement a queueing strategy. We maintain a separate queue for each neighboring interlay node. Each time a path through a next-hop neighbor becomes available for an additional packet (the criterion for this is described below), a packet is moved from its designated queue to a central active queue. Whenever the interface is ready to send another packet, packets are always taken from the head of the active queue. Routing packets and other high priority packets are placed in the active queue directly.

Two WINE nodes coordinate with each other only when traffic flows along a path that contains them both. *The goal of the coordination is to keep one and only one packet on the fly between two neighboring WINE nodes, so that intermediate mesh nodes will never interfere with each other.* To this end, a WINE node periodically probes neighboring WINE nodes to determine the round trip time (rtt), and control the aggregate rate (r) of transient traffic around one packet per $rtt/2$. We elaborate the way interlay node rate limits the transient traffic in Section 2.2 and then describe how WINE nodes discover their neighboring WINE nodes in Section 2.3.

2.2 Rate control

Probing packets between two WINE nodes are always sent along the same routes as ongoing traffic, ensuring that only WINE nodes on that path will get the packets. The rate is estimated by

$$r_{est} = \frac{p_{size} + \gamma}{(t_{now} - t_p)/2} \tag{2.1}$$

where r_{est} is the link rate estimate, p_{size} is the size of the probe packet received, t_{now} and t_p are the current and packet timestamps, and γ is the constant based on the per-hop overhead, e.g., RTS/CTS and MAC headers, multiplied by the hop count between these two WINE nodes. The estimated rate is calculated separately for each neighboring interlay

node. Finally, we apply a standard low-pass filter to the rate estimate to compensate for any sharp changes in rate measurement.

Since rate probing packets always follow the route of traffic through a node, under dynamic routing a routing change may force the probing packet along a route which does not find a WINE node. In this case, an interlay node will eventually cease to send probing packets and stop limiting the rate on that path if it does not get a response within a timeout period. Without this mechanism in place, since rate limits are associated with next hops, we may be limiting the rate of a path below what it can support.

As we described earlier, as a packet comes down from the routing layer, it is added to a priority queue based on the next interlay node in its route. If a packet was last sent to this neighbor more than $\frac{p_{size}}{r_{est}}$ seconds ago, then we immediately move the packet to the active queue where it sits until the interface is able to send. Otherwise, the packet remains on the neighbor queue until the wait time has expired and the path is considered free. Then the packet is moved to the active queue. The active queue should never contain more than one packet destined for the same neighbor.

By waiting $\frac{p_{size}}{r_{est}}$ seconds after sending a packet p before sending another packet to that same neighbor, we give each packet enough time to reach another interlay node before adding additional traffic to the path. Because we place interlay nodes approximately at the boundary of the interference range of each other, as we will see in Section 3.3, we decrease the probability that consecutive packets following the same path will interfere.

2.3 Interlay node discovery

We now address the problem of how an interlay node can determine which nodes are neighboring interlay nodes. An interlay node is considered a neighboring interlay node if there is some traffic path that passes through both nodes and there are no other interlay nodes in between on the path. As traffic passes through an interlay node, it inserts its address into

the interlay header. When an interlay node receives a packet, it checks the interlay header to see if this field is filled. If so, it knows that this packet passed through an interlay node recently as well as the address of the interlay node. The interlay node then begins probing the discovered neighbor. If no traffic arrives with that node's address in the interlay header after a certain amount of time (2 - 4 seconds), we stop probing. In this way, the interlay nodes are activated on-demand to the transient traffic.

Chapter 3

Interlay Placement

Heavy interference in an unregulated 802.11 wireless mesh leads to large amount of packet drops, compromising the transport capacity of the network. Since the interlay node rate control ensures one and only one packet on the fly between two neighboring interlay nodes, the interference among all intermediate hops are eliminated. However, limiting the number of packet on the fly between two arbitrary mesh nodes may leave un-interfered link idle, compromising the transport capacity of the network as well. Therefore, the interlay nodes have to be carefully deployed to tradeoff between the above two scenarios to optimize the data transport. In this section, we first describe our model and definition of transmission capacity. We then describe a centralized optimal algorithm that maximizes the minimum transmission capacity of any area of the network. We finally describe a practical interlay deployment strategy that approximates the optimum.

3.1 Model and Notation

This section presents a model for the transmission capacity of a multihop network with interlay nodes based on the link interference graph and connectivity graph of the network. We use this model to develop an algorithm for building the wireless interlay network.

Definition: The *connectivity graph* $G = (V, E)$ of the wireless network is a graph where V is the set of nodes of the network and $(v_i, v_j) \in E$ if v_j is within broadcast range of v_i. We call the members of E links in the network.

Definition: The *link interference matrix* I is an $E \times E$ matrix. An entry $I(i,j)$ is the probability that link i is able to transmit when it contends with link j.

We assume that each link has an interference range r_i so that for any links outside of this interference radius, $I(i,j) = 1$. We also assume that the number of links within interference range is bounded by a density constant D for all links.

Definition: An *interlay partition* is a disjoint set of *interlay areas* where each interlay area $R = \{l_1, \ldots, l_n\}$ a set of n connected links such that for any link $m \notin R$, and $\forall l \in R$ a path from l to m must cross an interlay node.

Since our interlay node rate limiter works to eliminate interference within an interlay area, we assume that for a pair of links i and j within the same interlay area, $I(i,j) = I(j,i) = 1$.

Definition: The *transmission capacity* c_l of a link l is the frequency with which it successfully contends for the channel. This is the rate at which it attempts to contend for the channel, $\frac{1}{R_l}$, where R_l is the size of the interlay area containing l, times the probability that it can transmit when it contends for the channel. Thus, the capacity of a link is given by

$$c_l = \frac{1}{R_1 \ldots R_n} \sum_{\substack{r_1 \in R_1 \\ r_2 \in R_2 \\ \ldots \\ r_n \in R_n}} \prod_{i=1}^{n} I(l, r_i) \tag{3.1}$$

where $R_1 \ldots R_n$ is the set of interlay areas in the network. This assumes that exactly one link is active in each interlay area at a time. Thus, each link will contend with at most n links simultaneously. The above equation simply sums the probability of successful transmission over every possible combination of active links in the n interlay areas.

Definition: The *contention-free transmission capacity* c_R of an interlay area R is the minimum transmission capacity of its links.

3.2 Maxmin interlay node placement

In this section, we present an interlay node placement algorithm which achieves maxmin fair link capacities. We wish to maximize the minimum capacity of any interlay area. However, this is the same as maximizing the minimum capacity of any link. First, we prove the following lemma:

Lemma 1 *Let $l_1 \ldots l_n$ be a set of n links in distinct interlay areas. If there is a link l_i such that $\prod_{j \neq i} I(i,j) < \frac{1}{n}$ then the capacity c_{l_i} would be higher if $l_1 \ldots l_n$ were in the same interlay area.*

Proof: Suppose links $l_1 \ldots l_n$ were in the same interlay area. Then the capacity of link l_i would decrease by a factor of n, since the denominator in the capacity equation would decrease by a factor of n. However, the numerator in the capacity equation would increase by a factor $> n$, since we eliminate a term $\prod_{j \neq i} I(i,j)$, from the interference product which is $< \frac{1}{n}$.■

The above Lemma suggests an algorithm that maximizes the minimum interlay area capacity in a wireless mesh. First, we find a minimum capacity link and optimizes its capacity by adding links to new interlay area if this will increase that link's capacity. This is repeated as each new minimum capacity link is found until it is impossible to increase the throughput of the minimum capacity link. During this process, links may be added to multiple interlay areas. If a link is found which needs to be in multiple interlay areas, it is placed in whichever area it will minimize the maximum capacity link. The pseudo codes of the algorithm is shown in Algorithm 1.

Correctness: Lemma 1 characterizes when a set of links should be in the same interlay area. At each iteration of the outer loop, Algorithm 1 checks each connected subset within interference range of the minimum capacity link and creates a new interlay area if doing so will maxmize that link's capacity. If no new interlay area can improve the capacity, then the minimum capacity link is already at a maximum and the algorithm terminates. Each link

Algorithm 1: Maxmin interlay node placement

for each link with current minimum capacity **do**
  for each connected subset S of links within interference range of min capacity link **do**
    if $\prod_{i \neq l_{min} \in S} I(l_{min}, l_i) < \frac{1}{S}$ **then**
      create a new interlay area containing links in S
      if one of these links is already present in an interlay area **then**
        compare the capacities of the min capacity link in each area with and without this link
        add link to the area which maximizes the minimum capacity
      end if
      update capacities of all links
    end if
  end for
end for

with minimum capacity is maximized over all possible local improvements. Since only local creation of interlay areas can improve a link's capacity (because of the interference range limit), every link we consider is at its maximum capacity, except when it is necessary to decrease that capacity for another link with less capacity. Thus, the minimum capacity of any given link is maximized by our algorithm.

Complexity: Since each link is considered at most once, the outer loop may run E times. Checking every connected subset within interference range takes at most $K = 2^D$, where D is the upper bound of the number of links within a local interference range. Updating the capacity of each link, if necessary, takes E time. Thus, the total running time of this algorithm is $O(KE^2)$.

3.3 Simplifications

While Lemma 1 leads to the development of Algorithm 1 that maximizes the minimum interlay area capacity, it requires the global link interference matrix I as one critical input. Although pioneering mesh interference measurements [5] have shown that the interference relations are stable for days, the measurement itself is disruptive to the normal operation of

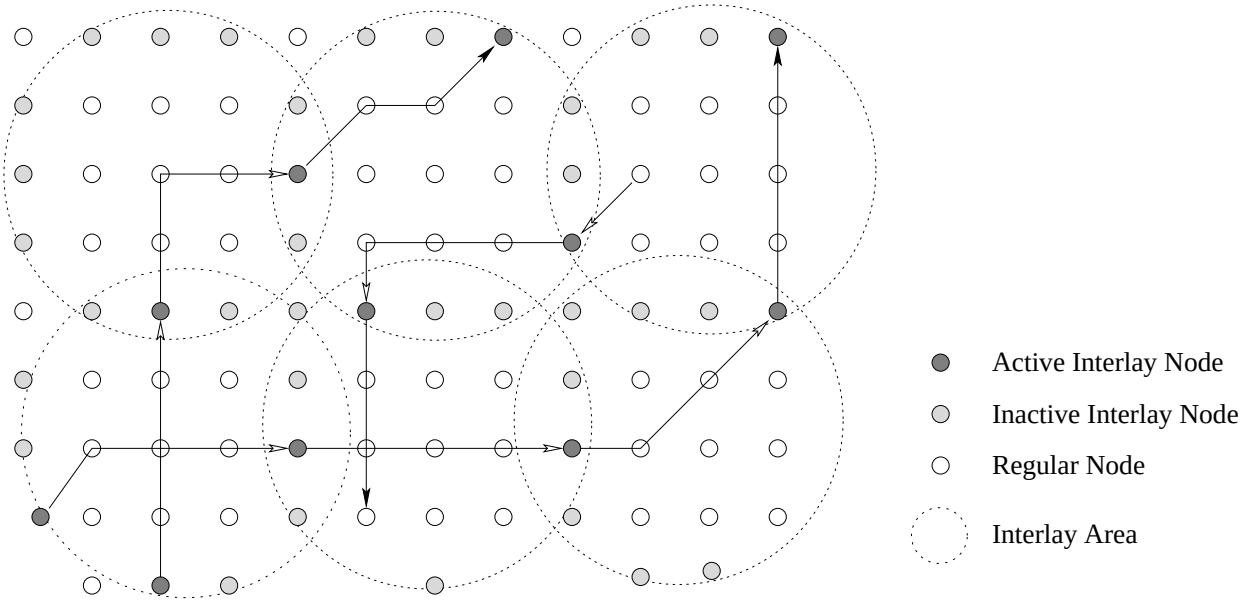

Figure 3.1: An example of a simplified interlay node deployment on a grid topology.

the network. In this section, we make some simplifying assumptions regarding the layout of the interference matrix. Then, we show how with these simplifications we match the intuition that interlay nodes should be placed at the interference boundary of highly interfered wireless links. This allows us to heuristically deploy interlay nodes without the need for an algorithm requiring an interference matrix, as long as we accept that the interlay node deployment will no longer be optimal.

A popular assumption about interference characteristics in wireless mesh topologies is the "2-hop interference metric." This is a generalization from the hidden terminal problem, where it is observed that nodes two hops away from each other will not be able to receive RTS or CTS. Thus, they will attempt to transmit to nodes which are blocked by other transmissions. Under this model, we assume that links within two hops of a given link will have a high rate of interference with the given link. Also, neighboring links will interfere, but not as much.

Examine a single link l with capacity c_l as defined in the previous section with k neighboring links and m two-hop neighboring links. If l has $I(l, l') = x$ with neighboring links

l' and rate $I(l, l'') = y$ with two-hop neighbors l'', then from 3.1 our algorithm creates an interlay area containing l and its 1 and 2 hop neighbors as long as

$$\frac{1}{k+m+1} > x^k y^m, \tag{3.2}$$

since $c_l = \prod_{i=1}^{n} I(l, r_i) = x^k y^m$ without the interlay area, but $c_l = \frac{1}{R_l} = \frac{1}{k+m+1}$ when all the $k+m$ links are in the same interlay area.

For example, consider a grid topology and a two-hop interference metric so that $I(l, l') = 0.5$ if l' is within two-hops of l, and $I(l, l') = 1$ otherwise. Then, applying the equation from above gives $\frac{1}{23} > \frac{1}{2}^{22}$, which implies that these links should be in the same interlay area to maximize the capacity of link l. This implies that by applying this algorithm to a grid, we should see interlay areas containing low-capacity links and their two-hop neighbors. This confirms the intuition driving the interlay design - that nodes placed just outside of the interference range of one another are better able to coordinate to reduce interference than nodes placed at the ends of the network.

This leads us to a simple heuristic for practical interlay node placement: simply *place interlay nodes so that they are at most two hops apart from the center of circular interlay areas.* For an example of this type of deployment, see figure 3.1. This example uses the common two-hop interference heuristic. We have implemented and run our optimal deployment algorithm on the grid topology and produced a similar arrangement of interlay regions. All the nodes on the boundary of an interlay area become interlay nodes. Any path that crosses more than one interference area will pass through interlay nodes. Those interlay nodes will then be *activated* and start to probe each other to enforce a rate-limit on that path.

Chapter 4

Interference Measurement

Our method for static allocation of interlay nodes requires knowledge of the interference matrix of the network. This section discusses our efficient, non-disruptivemethod for measuring interference between pairs of links in a wireless mesh network.

There is already ongoing research in this area. For example, [30] has developed a method for approximately, scalably measuring interference. Additionally, they showed that interference changes in a static network at a timescale on the order of days. Thus, we believe that the interference relationship among links will not change fast enough to affect any dynamic method of interlay node placement. Our method improves on that of [30] since we do not require that the network be shut down before measurement can take place.

Following our definition of the interference matrix in section 3, we seek to measure the probability that a transmission on a link is successful when another link is transmitting simultaneously. This is the same property as measured in [30].

4.1 Method

We measure the interference matrix by monitoring and processing network traffic files created by the tcpdump [35] utility. The latest version of tcpdump, 3.9.5, in conjunction with the madwifi-ng driver for Atheros 802.11 chipsets, supports the IEEE Radiotap headers present in 802.11 traffic. For details about our physical network deployment, see section 6.

The radiotap headers contain the MAC addresses of both the source and destination nodes , a time synchronization function with microsecond resolution, the data rate, channel,

and signal strength for each transmission. For an example of tcpdump's output with these features enabled, see listing 4.1.

Listing 4.1: Tcpdump output

```
12:48:42.983597 18446741804742041265us tsft short preamble
11.0 Mb/s 2462 MHz (0x0480) antenna 1 6dB signal
DA:00:02:6f:21:0a:b0 SA:00:02:6f:20:ed:41 BSSID:02:02:6f:20:b2:3b
LLC, dsap SNAP (0xaa), ssap SNAP (0xaa), cmd 0x03, IP
10.0.237.65.65532 > 10.0.10.176.22: S 2001621635:2001621635(0)
win 32768 <[|tcp]>

12:48:42.984588 short preamble 5.5 Mb/s 2462 MHz (0x0480) 0dBm
tx power antenna 1 DA:00:02:6f:20:ed:41 SA:00:02:6f:21:0a:b0
BSSID:02:02:6f:20:b2:3b LLC, dsap SNAP (0xaa), ssap SNAP (0xaa),
cmd 0x03, IP 10.0.10.176.22 > 10.0.237.65.65532:
S 3259591274:3259591274(0) ack 2001621636 win 32768
<mss 1460,nop,wscale 0,[|tcp]>

12:48:42.985976 18446741804742043652us tsft short preamble
11.0 Mb/s 2462 MHz (0x0480) antenna 1 9dB signal
DA:00:02:6f:21:0a:b0 SA:00:02:6f:20:ed:41
BSSID:02:02:6f:20:b2:3b LLC, dsap SNAP (0xaa), ssap SNAP (0xaa),
cmd 0x03, IP 10.0.237.65.65532 > 10.0.10.176.22:
S 2001621635:2001621635(0) win 32768 <[|tcp]>
```

The timestamp available from the TSF is much more accurate than that available from the operating system. We measured the accuracy of the TSF timestamps by comparing the recorded timestamps of a pair of nodes when they receive a packet transmitted by a third

node. We found that the error in timestamps was less than 100us. The tcpdump output is well-suited for measuring interference at packet-level time granularity.

From this data, we can calculate the relative transmission rate of each link with respect to each other link within range by comparing its successful transmission rate when the other link is and is not transmitting. If A is the event of a successful transmission on a link in a time period, and B is the event of an attempted transmission on the other link in that same time period, then we can calculate the conditional probability $P(A \mid B)$ from $P(A \wedge B)$ and $P(B)$ by the definition of conditional probability. We can easily compute $P(B)$ and $P(A \wedge B)$ (by identifying those instances where only A and B occur) from the tcpdump data.

To smooth out large fluctuations in interference rate, we report the average success and failure of packet transmissions on each link for the last t seconds.

One limitation of this method is that it has no way of predicting interference between pairs of links which do not have transmissions in the same time period. However, if a pair of nodes contend for the channel simultaneously, then by definition they do not interfere in any practical way. For example, a link which is active never or rarely might not appear to interfere with any other links, simply because it was never active during other transmissions. In networks with very low throughput or traffic rates, it may be difficult to observe any interference at all. However, for practical applications this is no real loss.

4.2 Software Design

We'll now describe a proposed software architecture for an interference measurement agent. Each node will cache its tcpdump log until it reaches a certain size. Then it compresses and transmits the data to a interference measurement service. This service parses the traffic logs, identifies links by source and destination MAC address, and calculates interference between those links based on the received data. The interference matrix is now available for query

by any node. Since interference relationships are stable on the order of days, this sort of out-of-band processing of the tcpdump data allows for minimal measurement and processing overhead for network nodes.

We have implemented a prototype version of this software in Python. The tcpdump logs must be collected manually or using other tools, but the software will produce from a set of tcpdump logs a set of link inteference measurements. We have successfully tested our software on data collected from the Champaign-Urbana community wireless network [2] and demonstrated interference relationships between two pairs of linke on this 6 node network.

Chapter 5

Simulation Evaluation

We now evaluate the performance of WINE together with our proposed WINE node placement algorithm. We have implemented the WINE architecture in the ns2 simulator. First, we examine the improvement that the appearance of a more resilient link mechanism can give to to the upper-layer transport protocols, including TCP and real-time protocols. We then demonstrate the performance benefits of WINE to the routing protocols in terms of route stability.

Some parameters are global to all the simulations. The MAC layer in all cases was standard 802.11b. The data rate is set to 11 Mbps, with a 2Mbps basic rate. When we test TCP, we always use TCP NewReno implementation. Until section 5.3, in which we evaluate the performance of DSR with WINE, our simulations use a static routing assignment to isolate WINE from the effects of routing protocols. Nodes are always placed 240 meters from their neighbors. The broadcast range is set to 250 meters and the interference range is 550 meters.

5.1 Bulk Data Transfer

Our first simulation topology is described in figure 5.1, an 8 hop linear chain. Nodes 0, 3, 6, and 8 are chosen as WINE nodes by our WINE node selection algorithm in Section 3.2. The interference matrix was populated based on the model proposed in [19]. We simulate ftp transfers over NewReno TCP from node 0 to node 8. Each flow starts 20 sec after the previous one and runs for 100 seconds. The total simulation time was 200 sec.

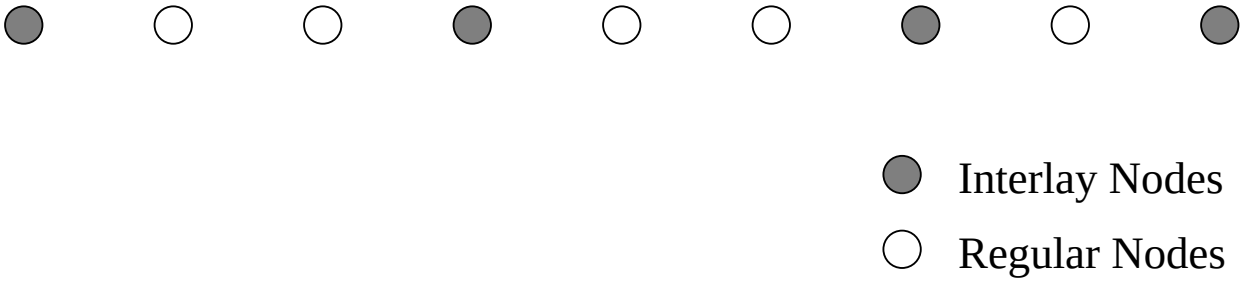

Figure 5.1: A wireless interlay on a linear topology. Darker nodes are WINE nodes.

Num of flows	Bytes received		Packets dropped	
	Basic TCP	TCP w/ WINE	Basic TCP	TCP w/WINE
1	4.60 MB	4.99 MB MB	31	1
2	5.55 MB	6.01 MB	75	4
3	6.55 MB	7.03 MB	107	2
4	7.41 MB	8.06 MB	117	2
5	8.32 MB	9.01 MB	147	9
6	9.23 MB	10.04 MB	173	5
7	9.34 MB	10.03 MB	179	5
8	9.32 MB	10.03 MB	200	4

Table 5.1: Throughput and packet loss in multiple simultaneous TCP flows on a linear topology

TCP throughput is improved by approximately 15% in each simulation. This is unsurprising given our much lower packet loss rate as shown in table 5.1. Overall, we decrease packet loss by 90-95%. The throughput gains from the decreased packet loss are lower than expected, since, as was shown in [18], on a wireless network, TCP grows its window size beyond the networks capacity, and packets remained queued at all nodes even under heavy losses. Because WINE rate limits TCP independently of the window size, it forces more packets into the queues. Since the nodes within an interlay region no longer compete as heavily for the shared medium, packet collisions and losses are much less likely. This limits the number of timeouts that TCP experiences, improving total throughput performance.

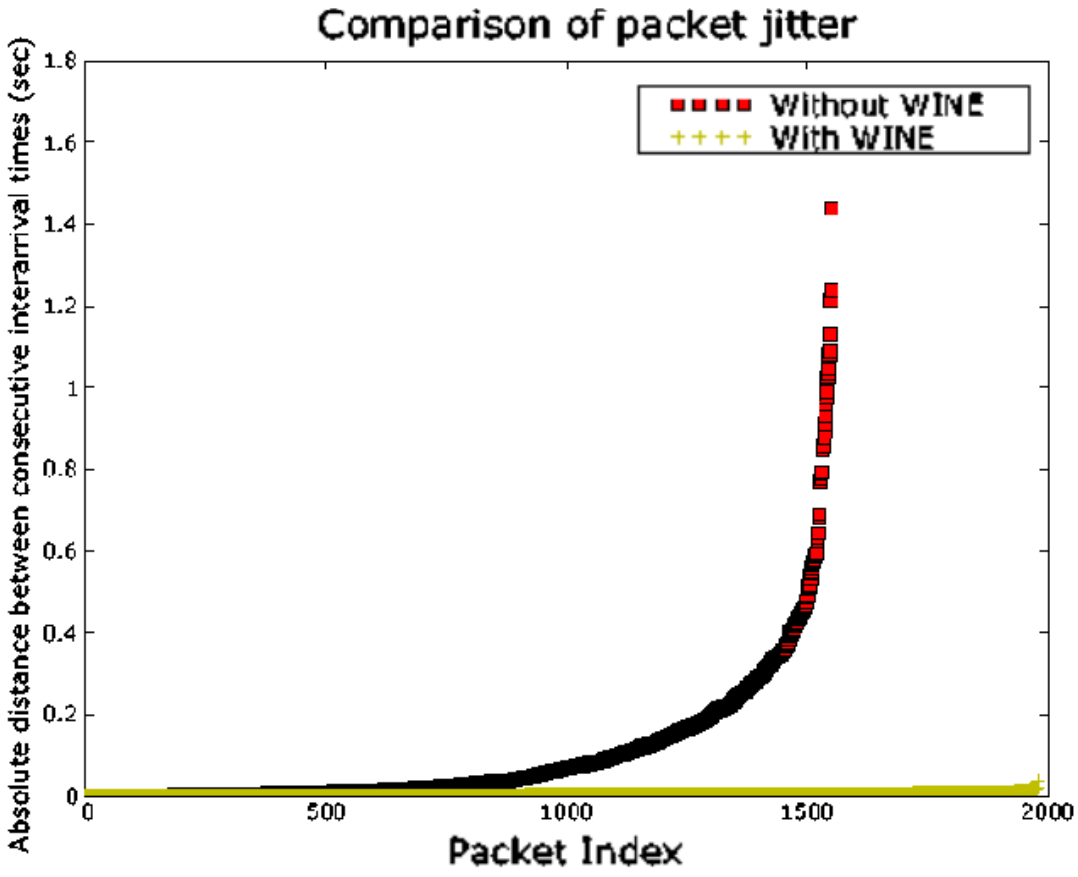

Figure 5.2: Delay jitter w/ and w/o WINE.

Next, we demonstrate how WINE can improve the delay variation of bulk data traffic.

Since we decrease contention along all WINE-managed paths, there is less variation in the RTT between source and destination. We measured the delay variation, as defined in [14], which defines jitter as the absolute difference in consecutive packet interarrival times, i.e.

$$abs((t_1 - t_2) - (t_2 - t_3)) \tag{5.1}$$

where t_1, t_2, and t_3 are the arrival times of three consecutive packets.

We use a constant bit rate UDP flow along the linear topology set to 2.66 Mbps. Average delay variation decreased from 100 ms to 3 ms, while 95^{th} percentile delay variation decreased from 420 ms to 9 ms. Figure 5.2 gives a broad overview of the results.

Our second simulation topology is a cross topology as in figure 5.3 with a 9-node horizontal path and a 9-node vertical path. WINE interlay nodes are deployed according to the strategy of section 3 and shown in figure 5.3. We simulate one ftp flow along each path..

For this topology, unmodified TCP traffic achieves more than 90% utilization of the capacity of the congested center region, thus WINE does not show much improvement on throughput. However, the addition of WINE interlay nodes decreases the packet loss rate by 70% for this topology. This result demonstrates that WINE nodes which explicitly coordinate only pairwise can nevertheless implicitly coordinate to effectively regulate the traffic in interlay regions as well as on simple abstract links. Again, a lower packet loss rate does not translate to significant throughput gains. The reason is that, as shown in [18], TCP grows its window overly large and enough packets are queued at each node. Even when TCP cuts its congestion window, the window is still large enough to fill the pipe of the links.

By transmitting at a rate which limits contention within interlay regions in the WINE network, WINE is able to greatly reduce the frequency of packet drops due to expired 802.11 retransmission timers. This improves network utilization both because network resources will not be wasted on packets which are eventually dropped, but also by eliminating spurious TCP timeouts. These improvements come without any modification to the underlying 802.11

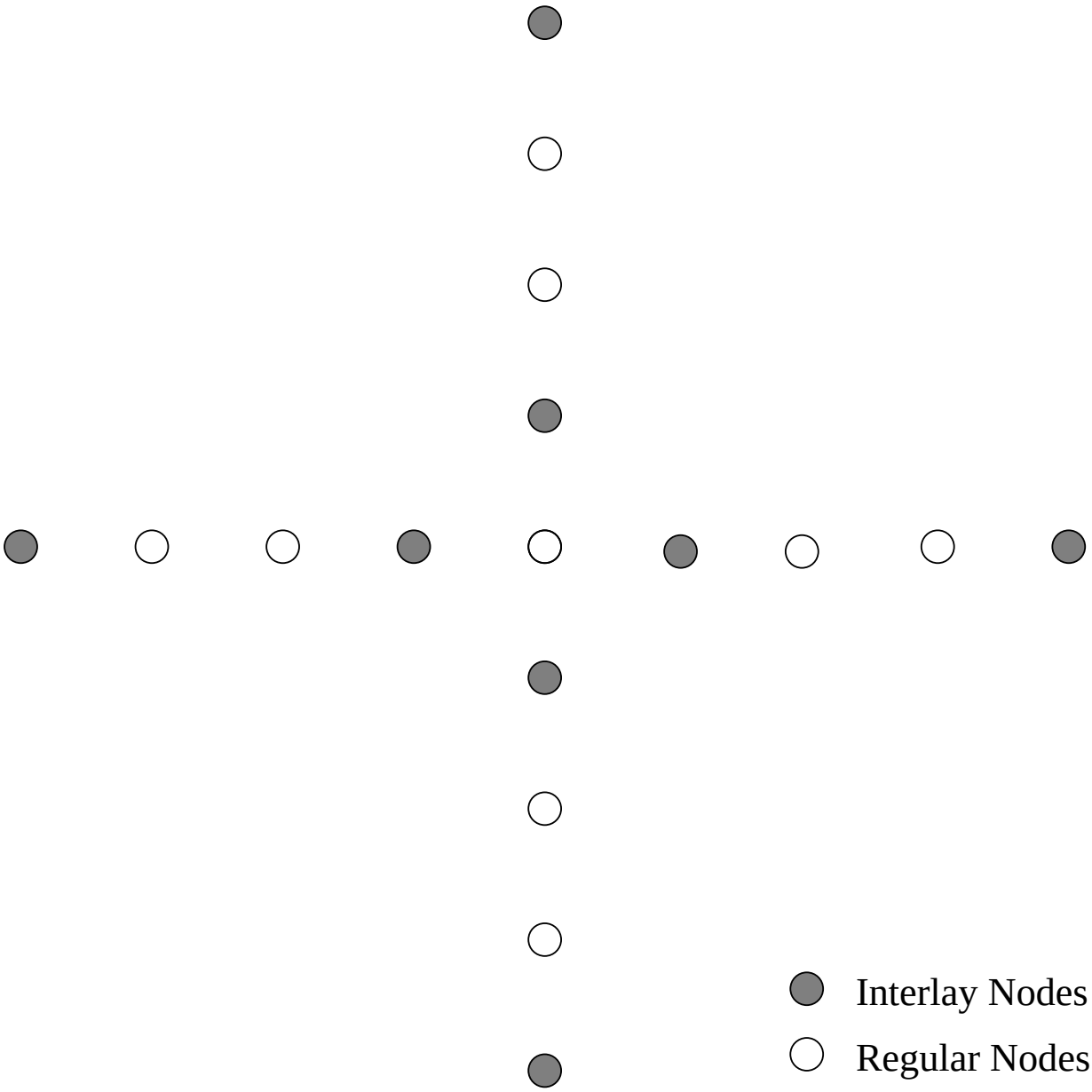

Figure 5.3: Crossing topology. Darker nodes represent WINE nodes.

DCF protocol or the upper-layer TCP protocol.

5.2 Real-time Traffic

In this section, we study the performance of a multihop wireless mesh in simulation with real-time traffic load, both with and without the WINE interlay. We use the linear topology

Video trace file	Bytes received		Packets dropped	
	Basic UDP	UDP w/ WINE	Basic UDP	UDP w/WINE
Aladdin	179.78 MB	612.67 MB	2697	933
Silence	358.34 MB	802.49 MB	3063	805
Star Wars	437.19 MB	881.47 MB	19	15

Table 5.2: Throughput and packet loss in real-time video trace files

of figure 5.1. We simulate H.263 encoded video transfer over UDP using trace files from [17]. Two video streams were transmitted simultaneously in order to create a nontrivial load on the network.

As table 5.2 shows, WINE increases throughput by approximately 100% while decreasing contention-based packet loss by 75%. The *Star Wars* video trace has very little packet loss both with and without WINE.

WINE also reduces jitter, an important metric for real-time playback of the streaming video. In the *Silence of the Lambs* video trace, for example, WINE decreases the mean delay variation from 28 ms to 23 ms. The 95^{th} percentile delay variation is decreased from 84 ms to 71 ms. Overall, WINE reduces the jitter by approximately 15%.

Next, we examine the performance of real-time traffic in conjunction with a bulk-data traffic load. We simulate an FTP flow over TCP along with SSH traffic on the linear topology. All other variables are consistent with the other simulation. With WINE, throughput for the SSH flow is increased by more than 90% and packet loss for both flows is less than 1%. Packet loss for the SSH flow without WINE is high - more than 50% - and throughput is near zero.

WINE provides several benefits to real-time traffic. By decreasing total packet losses, higher throughput is possible, which should result in higher video and speech fidelity. WINE also provides a more stable network which results in less jitter, providing a better experience of interactivity to the end-user. Also, since WINE sits below the transport layer, it is fully compatible with and can provide benefits to real-time transport protocols

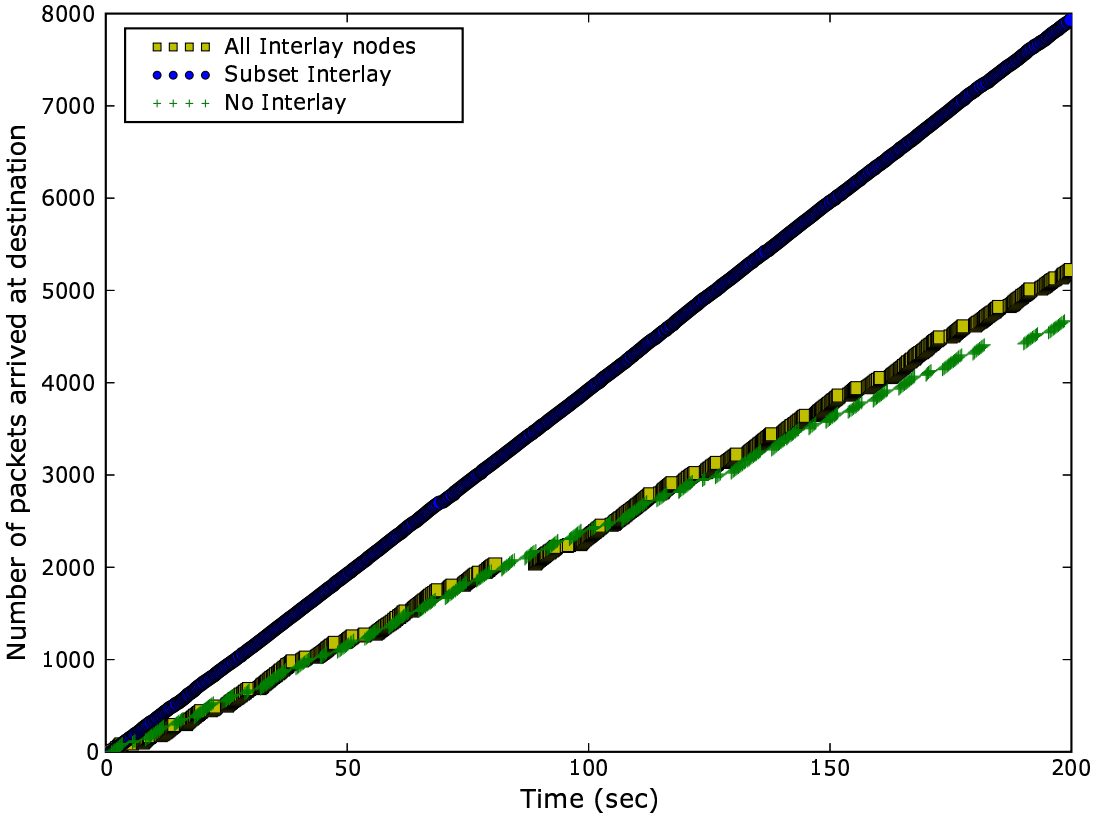

Figure 5.4: Throughput comparison of WINE enabled on all nodes, enabled only according to the interlay node selection algorithm, and a native network

Finally, to emphasize the point that not every node in the network should participate in the interlay, we run an additional experiment with every node acting as a WINE node. When WINE is enabled on every node, each node coordinates only with its neighbors to limit the rate of traffic with them. However, rate-limiting on a single link is "short-sighted", and does not handle interference from beyond one-hop neighborhood well. Thus, we should not expect much of a performance difference when every node acts as an interlay node. Figure 5.4 compares the performance of three scenarios on the 9-node linear topology. The first enables all nodes with WINE, the second with only some as in the linear interlay node deployment scheme of the previous scenarios, and the last without WINE enabled. Network performance in terms of packet loss and throughput is comparable between the scenario with all WINE nodes and with none. Neither approaches the performance of the case with only a subset of WINE nodes enabled.

5.3 Routing Protocol

In this section, we examine the performance of WINE with DSR, a popular wireless network routing algorithm. We have already demonstrated that WINE offers benefits in user level metrics such as jitter and packet loss rate, but now we show that it also improves routing layer performance by reducing route thrashing and spurious route breakages. Note that WINE deployment is independent from the traffic distribution or the routing protocol. WINE nodes are activated on demand as traffic flows through, along the route that determined by the routing.

For this examination, we experiment with two different network topologies. First, we simulate a simple 9 node linear chain topology to see the effect of WINE on routing protocols when route choice is not a consideration. Next, we simulate a 9 by 9 grid of nodes. For the experiments using WINE, we deploy interlay nodes as shown in figure 3.1, according to the interlay node selection heuristic described in section 3.3. We examine scenarios involving

one flow on the network as well as multiple flows. For the TCP traffic simulations, we use an FTP transfer on top of NewReno TCP. For the UDP scenarios in this section, we use a constant bit rate traffic generator with 1500 byte packets sent 100 times a second, generating 1.2 Mbps of traffic.

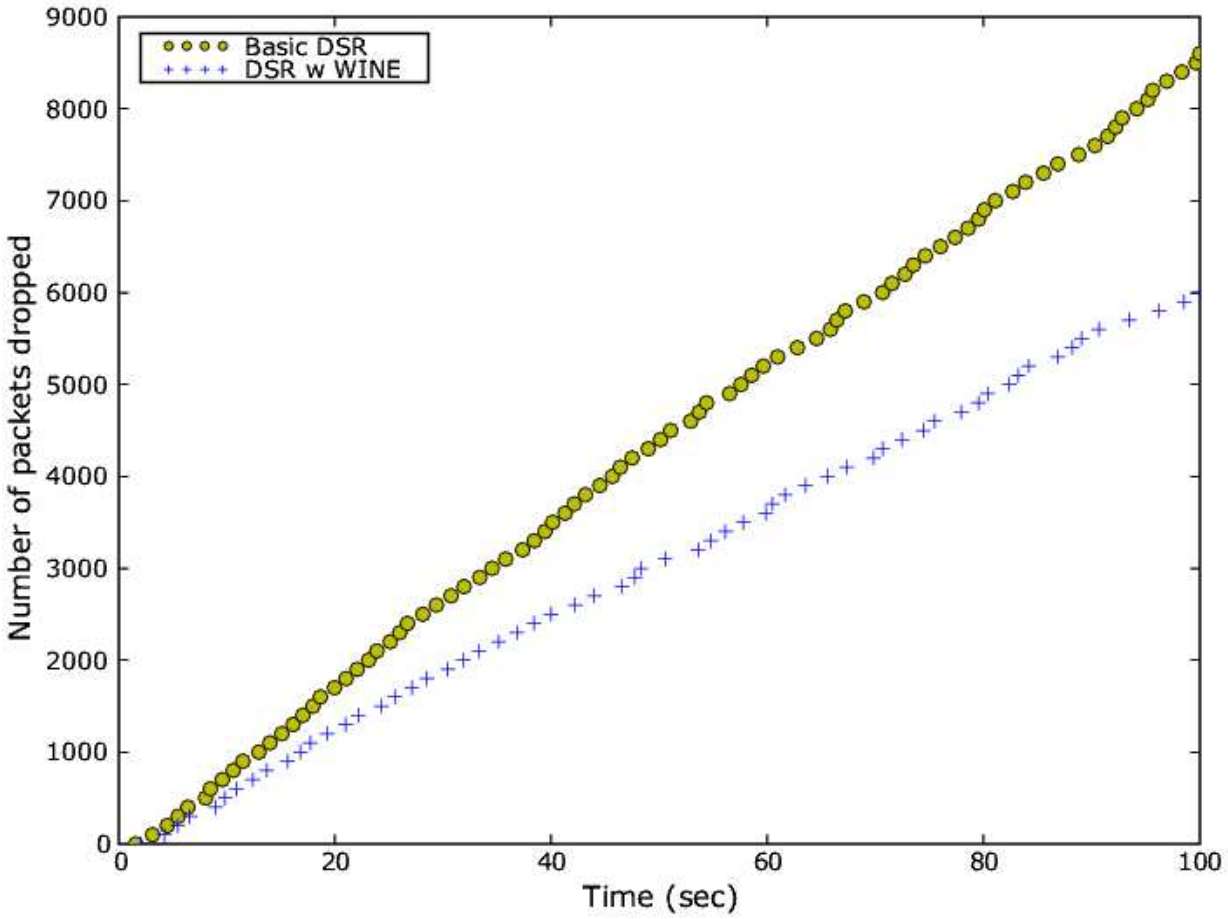

Figure 5.5: Comparison of packet loss rates between WINE and basic DSR with a single UDP flow

Our first simulation scenario tests a single TCP flow on both the linear topology and the grid. We seek to evaluate the performance of WINE in the absence of inter-flow interference. With WINE enabled on the linear path, the TCP throughput is 35% higher than with DSR

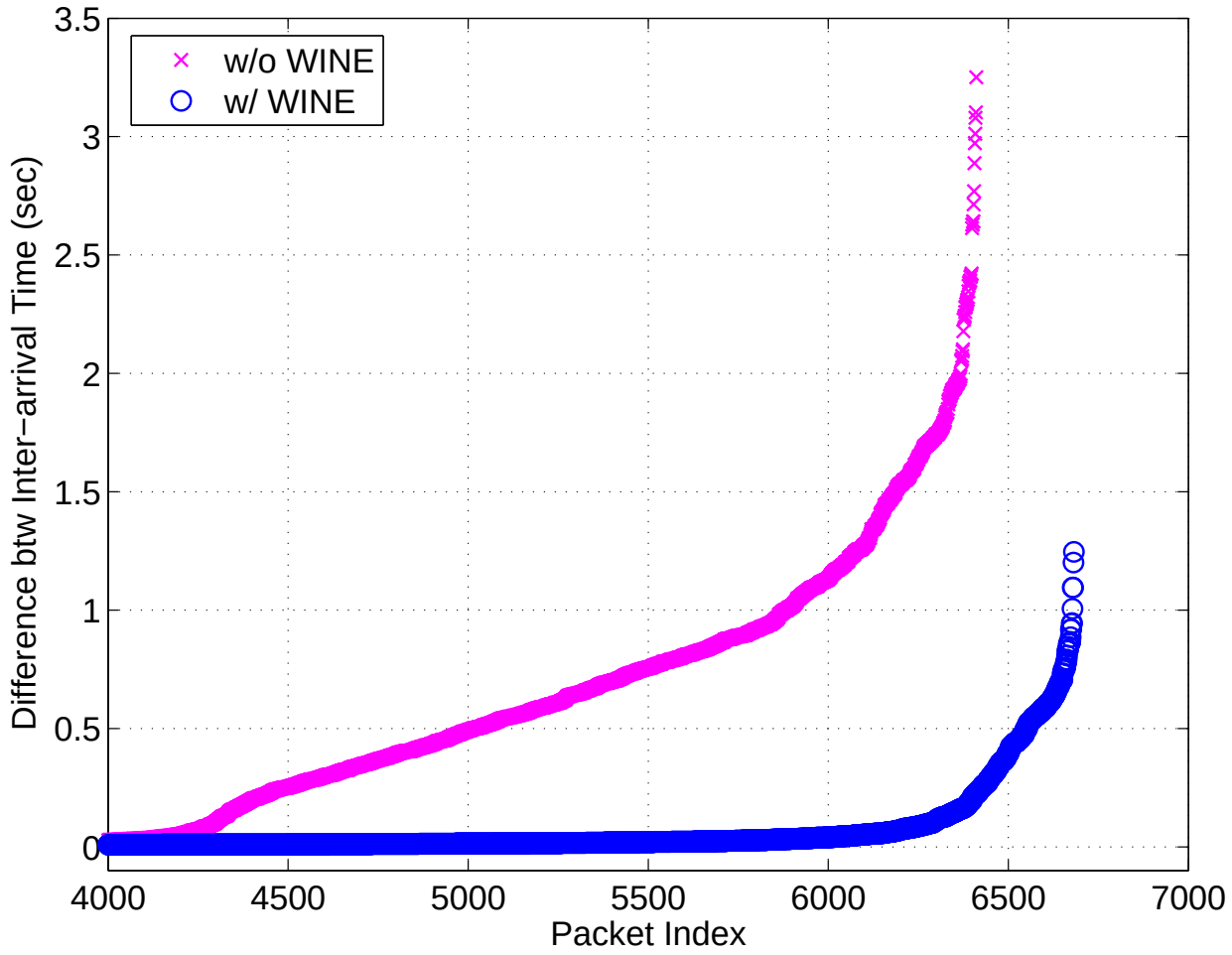

Figure 5.6: Comparison of absolute difference in interarrival times for a single flow on the grid topology

alone. This reflects the benefit of WINE with respect to lower packet loss rate (no packets are dropped in the WINE-enabled simulation), but also there is no throughput loss due to TCP timeouts from route breakdowns. Despite having only one route available on the path, DSR without WINE detects that the route has broken 389 times over the 200 second run of the simulation. The maximum time on a route is 20 seconds before the route is lost and must be rediscovered. In contrast, route breakages do not occur with WINE nodes enabled.

On the grid topology, we mimic the previous scenario by transmitting a single flow across the center of the grid. On the shortest path, traffic passes over 8 hops. While WINE improves

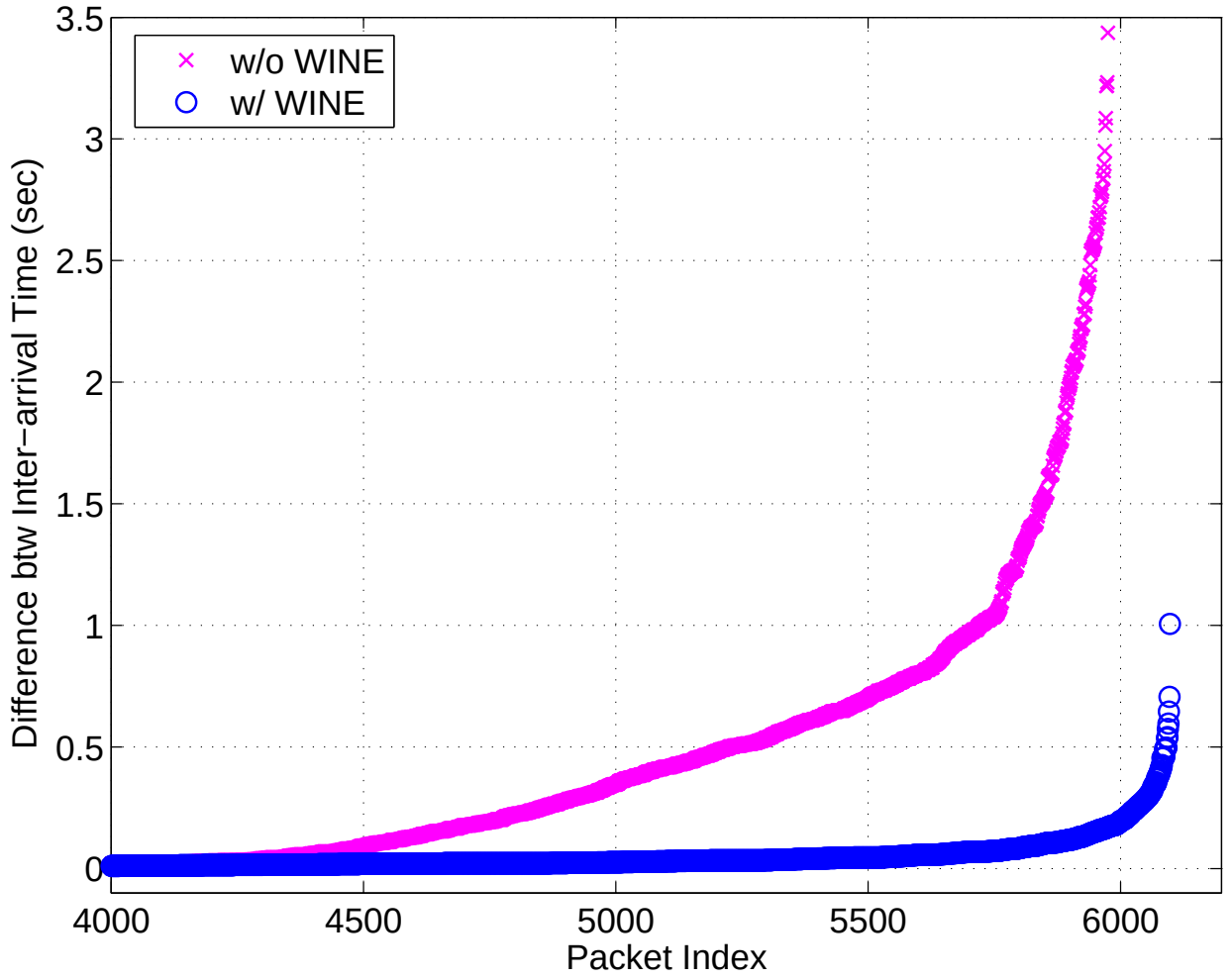

Figure 5.7: Comparison of absolute difference in interarrival times for a single flow among four random flows on the grid topology

the throughput only marginally, the packet loss rate is decreased by 67%. Additionally, the route is much more stable under WINE, changing 121 times with basic DSR, but stable on one route with WINE.

With the same scenario under UDP, WINE decreases packet loss by 30%. Packet loss by time is shown in figure 5.5. WINE reduces the mean interarrival time from 264 ms to 33 ms, while the 95-percentile interarrival time decreases from 1259 ms to 146 ms. Absolute difference between packet interarrival times are shown in figure 5.6 as a measure of the traffic

jitter.

Our final simulation scenario tests four flows with randomly chosen endpoints on the grid topology. As in the previous cases, we examine WINE performance with both TCP and UDP compared to basic DSR without WINE. In the TCP scenario, WINE cuts the average packet loss rate of the four flows by 48%. Average throughput improves only marginally, but flow fairness, as measured by the standard deviation of the throughput for the four flows, is improved by 30%.

When we consider UDP traffic instead, total throughput increases by 8% when WINE is enabled and the packet loss rate decreases by 8%. Route stability is drastically increased. With basic DSR, the route changes over 200 times between one pair of nodes, but when WINE is enabled the route stays on the shortest path throughout the simulation. Jitter shows improvements in this case as well. The absolute difference in packet interarrival times for one of the flows in this scenario are shown in figure 5.7. Mean interarrival time is decreased from 188 ms to 22 ms. 95-percentile arrival times is decreased from 962 ms to 86 ms with WINE.

We have demonstrated that WINE is compatible with DSR, a popular source routing protocol for wireless networks. In addition, WINE can provide benefits to the routing layer by reducing the number of packet losses due to interference that cause spurious route loss errors.

Chapter 6

Implementation

We have implemented our WINE architecture for the Linux operating system. In this chapter, we describe our testbed, discuss our experimental method, and present the results of our experiments. The results described here demonstrate that WINE provides some tangible benefits to existing wireless mesh networks.

6.1 Testbed

We have constructed a small wireless testbed with 7 Soekris Engineering net4801 network appliances. These have 266Mhz NSC Geode x86 compatible processors, 64 MB of RAM, and 512 MB of Compact Flash storage. Each device has a netgate miniPCI wireless card using the Atheros chipset. We configured the cards to use 802.11a MAC with autorate enabled.

The devices have been deployed in a linear topology on one floor of our office building (figure 6.1). Our building has a large number of 802.11g devices, thus we chose the 802.11a band to minimize extranetwork interference. The data rate was set to 12Mbps. All experiments were run at night to minimize human environmental variation. The link stability on the testbed was variable on the order of hours, requiring occasional changes to the node location. However, there is little variation in the link quality during our experimental runs. The maximum end-to-end rate recorded over the course of our network calibration was a little over 1 Mbps.

We installed Debian GNU/Linux with an unmodified 2.6.8 Linux kernel, using the default TCP configuration. This distribution includes selective acknowledgements by default.

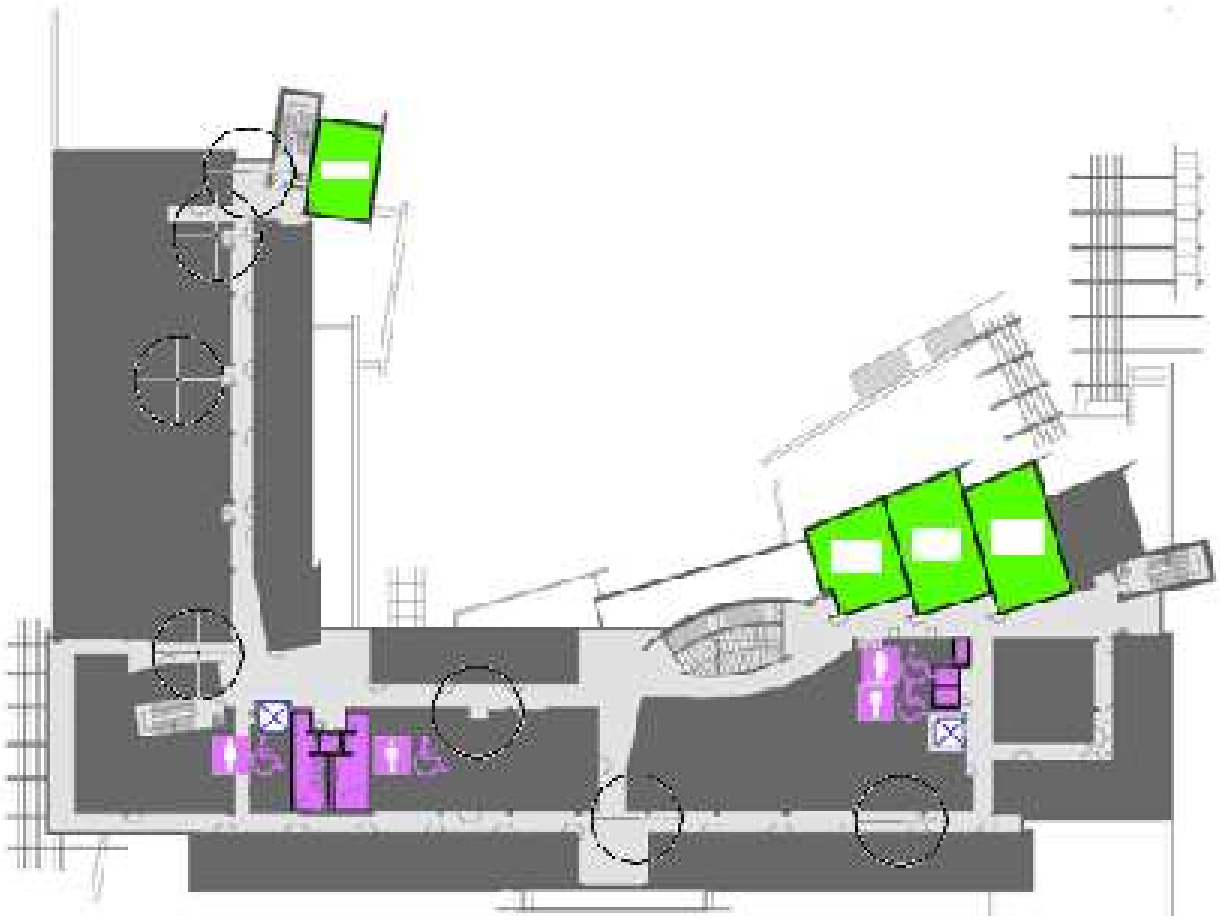

Figure 6.1: Map of the deployed testbed

The devices are configured manually with static routes. We have implemented the WINE interface queue for Linux 2.6 using the Linux kernel's ipqueue module. The ipqueue module allows one to dynamically queue packets in userspace.

The WINE software is implemented as a multithreaded userspace application linking to ipqueue. Figure 6.2 illustrates the architecture of our implementation. Each neighboring interlay node has a rate probing thread dedicated to it which sends probing packets at a fixed rate to determine the round trip time to that neighboring node. The rate probing thread reports its measurements to the queue manager. The queue manager maintains a queue for each neighboring interlay node. It releases packets from the queues to the ipqueue module at the rate calculated by the rate probing thread. The queue manager receives packets from

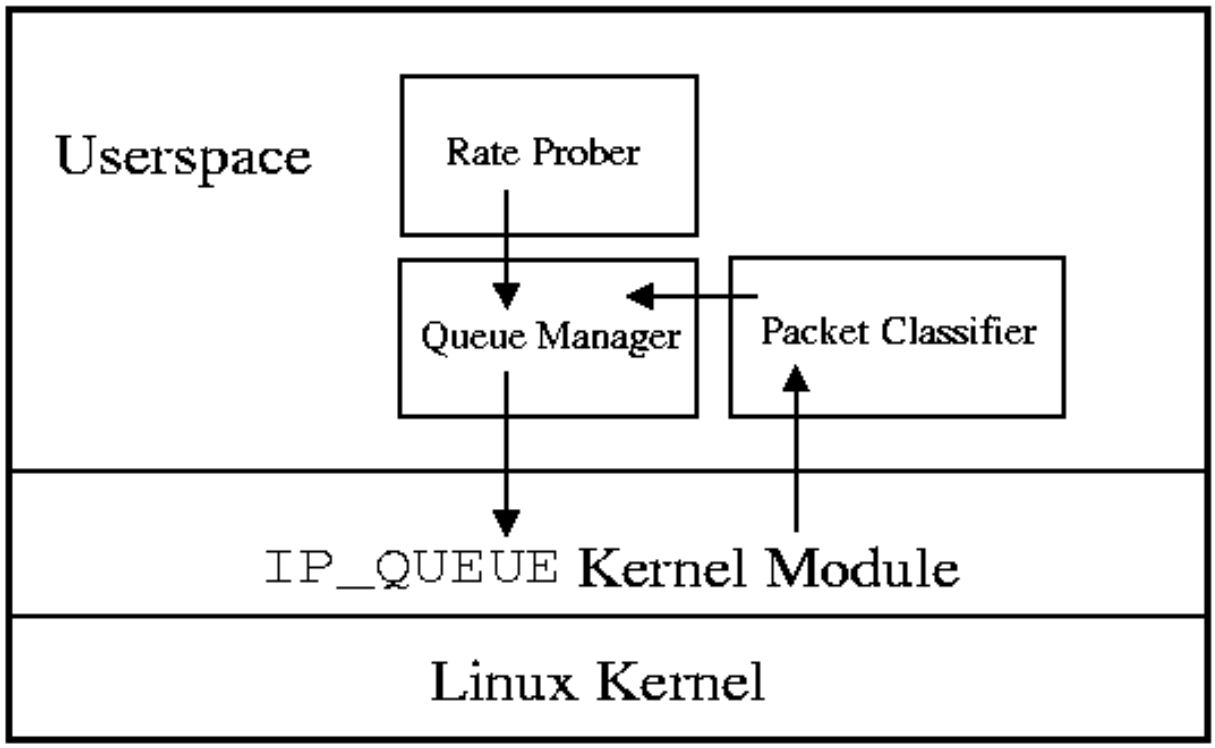

Figure 6.2: Architecture of a WINE interlay node

the packet classifying thread, which accepts packets from the kernel via the ipqueue module and classifies them according the next interlay node on their path. It then inserts the packets into the appropriate outgoing queue within the queue manager.

6.2 Experimental Method

We perform several experiments on our testbed. We wish to discover whether WINE offers the same performance benefits in a testbed as in exhibits in the simulation.

For the first experiment, we test the performance of WINE in each of the following scenarios. Before each test, we measure the packet loss rate of each link in our network using the flood option of the ping utility to ensure we have no poor quality links in the chain. Tcpdump is used to collect network traffic data at the source and destination. We activate WINE at the pre-chosen interlay nodes and start a single FTP flow from node 1

to node 7 transferring a 2.5MB binary file. When this completes, we collect the tcpdump trace file, restart the tcpdump utility, deactivate WINE, and transfer the file again, this time using TCP without WINE nodes enabled on the network. The overall time between WINE-activated experiment completion and the start of the native TCP experiment is less than 5 minutes. We repeat this process 5 times. Overall throughput is reported by the ftp application and confirmed by the tcpdump trace file. The packet loss rate is calculated by comparison of the trace files at the source and destination. Packets sent by the source but not received at the destination are counted as lost.

6.3 Results

Overall, total network throughput is slightly less with WINE enabled. Our testbed was not plagued with poor TCP performance as is often reported in the literature. We conclude that due to the linearity of our network topology, actual link interference was often less than is typical in existent networks. On average, we decrease total TCP throughput by about 6%, but we also reduce total TCP packet loss by 22%.

6.4 Discussion

There are several reasons why our real-life performance lagged so much behind our simulated performance. One significant deviation was that the naked performance of our testbed was much better than predicted by the simulator. While the packet loss rate of the simulator was greater than 90%, our testbed, when calibrated perfectly, had a packet loss rate of ¡ 1%, close to the level of a wired network. Actual interference between links at two-hops was negligible. Without interference causing large packet loss in the network, throughput was very high - greater than 2.2 Mbps when running at 12 Mbps. We conclude that link quality is the primary cause of packet loss in real networks, so much so that interference is negligible

in comparison. When we carefully calibrated our links so that the individual packet loss rate was 0, performance problems in the network as a whole virtually disappeared.

Nevertheless, WINE demonstrated that it is effective at providing interference regions in which only a small number of packets are competing for bandwidth at any given time. It does this without reducing the effective throughput of the network.

Chapter 7

Conclusion

We have presented a new architecture for wireless mesh networks that attempts to solve the problem of interference that appears over multiple hops. It does this by deploying a system of interlay nodes on the network. These nodes communicate with each other to reduce the traffic rate between them, and thus the interference caused by the presence of too many packets on a path. To facilitate this architecture, we have also developed an algorithm which seeks to compute an optimal placement of interlay nodes to minimize the total interference on the network. We also gave an heuristic based on this algorithm for grid topologies, which greatly simplifies the process of choosing interlay nodes. We also provide a new method for measuring interference in a running network.

We tested our design first through simulation of a variety of network topologies and with popular wireless mesh routing algorithms. The results were promising, providing significant increases in throughput and decreases in packet loss for certain topologies. Also, we showed that WINE provides additional benefits to real-time traffic, such as SSH and video. Finally, we built a real test environment for our new protocol and implemented it in the Linux kernel. Real-life performance improvements were less significant.

7.1 Future work

In testing our protocol in our real-life testbed, we exposed several assumptions about wireless mesh networks, such as the all-or-nothing interference relationship of the ns-2 simulator, the poor performance of unmodified wireless mesh networks in the absence of high link failure

rates, and the adverse effect of TCP on popular wireless routing protocols. Future work would be advised to investigate other sources of poor performance in wireless mesh networks, such as variety in link quality first, and interference second.

To expand on our interference measurement technique, we could make the process of collecting logs automatic and provide a query protocol to provide an interference reporting service. With this facility enabled, other applications may be possible which require knowledge of the interference relationships on a wireless mesh network.

References

[1] 802.11s working group.

[2] Champaign-Urbana community wireless network. http://www.cuwireless.net/.

[3] MIT roofnet. http://www.pdos.lcs.mit.edu/roofnet/.

[4] MSR self-organizing neighborhood wireless mesh networks. http://www.research.microsoft.com/mesh/.

[5] S. Agarwal, J. Padhye, V. N. Padmanabhan, L. Qiu, A. Rao, and B. Zill. Measurement and estimation of link intreference in static multi-hop wireless networks. In *Proceedings of ACM IMC*, 2005.

[6] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proceedings of ACM SOSP*, 2001.

[7] L. Bao and J. Garcia-Luna-Aceves. A new approach to channel access scheduling for ad hoc networks. In *Proceedings of ACM MobiCom*, 2001.

[8] S. Basagni. Distributed clustering for ad hoc networks. In *Proceedings of International Symposium on Parallel Architectures, Algorithms, and Networks*, 1999.

[9] D. Berger, Z. Ye, P. Sinha, S. Krishnamurthy, M. Faloutsos, and S. K. Tripathi. TCP-friendly medium access control for ad-hoc wireless networks: Alleviating self-contention. In *Proceedings of IEEE MASS*, 2004.

[10] J. D. Camp, E. W. Knightly, and W. S. Reed. Developing and deploying multihop wireless networks for low-income communities. In *Poster of IEEE INFOCOM*, 2005.

[11] R. R. Choudhury, X. Yang, R. Ramanathan, and N. Vaidya. Medium access control in ad hoc networks using directional antennas. In *Proceedings of ACM MobiCom*, 2002.

[12] D. D. Clark. The design philosophy of the darpa internet protocols. In *Proceedings of ACM SIGCOMM*, 1988.

[13] D. D. Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *Proceedings of ACM MobiCom*, 2003.

[14] C. Demichelis and P. Chimento. RFC 3393 - ip packet delay variation metric for ip performance metrics (ippm). `http://www.faqs.org/rfcs/rfc3393.html`.

[15] R. Draves, J. Padhye, and B. Zill. Comparison of routing metrics for static multi-hop wireless networks. In *Proceedings of ACM SIGCOMM*, 2004.

[16] R. Draves, J. Padhye, and B. Zill. Routing in multi-radio, multi-hop wireless mesh networks. In *Proceedings of ACM MobiCom*, 2004.

[17] F. Fitzek, M. Zorzi, P. Seeling, and M. Reisslein. Video and audio trace files of pre–encoded video content for network performance measurements (extended version), 2003.

[18] Z. Fu, H. Luo, P. Zerfos, S. Lu, L. Zhang, and M. Gerla. The impact of multihop wireless channel on TCP performance. *IEEE Transactions on Mobile Computing*, 4(2):209–221, March/April 2005.

[19] M. Garetto, J. Shi, and E. W. Knightly. Modeling media access in embedded two-flow topologies of multihop wireless networks. In *Proceedings of ACM MobiCom*, 2005.

[20] Y. hua Chu, S. G. Rao, S. Seshan, and H. Zhang. Enabling conferencing applications on the internet using an overlay multicast architecture. In *Proceedings of ACM SIGCOMM*, 2001.

[21] K. Jain, J. Padhye, V. Padmanabhan, and L. Qiu. Impact of interference on multi-hop wireless network performance. In *Proceedings of ACM MobiCom*, 2003.

[22] R. Karrer, A. Sabharwal, and E. Knightly. Enabling large-scale wireless broadband: The case for TAPs. In *Proceedings of HotNets*, 2003.

[23] H. Luo, P. Medvedev, J. Cheng, and S. Lu. A self-coordinating approach to distributed fair queueing in ad hoc wireless networks. In *Proceedings of IEEE INFOCOM*, 2001.

[24] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang. TCP westwood: Bandwidth estimation for enhanced transport over wireless links. In *Proceedings of IEEE ICNP*, 2002.

[25] A. Nakao, L. Peterson, and A. Bavier. A routing underlay for overlay networks. In *Proceedings of ACM SIGCOMM*, 2003.

[26] B. Raman and K. Chebrolu. Design and evaluation of a new MAC protocol for long-distance 802.11 mesh networks. In *Proceedings of ACM MobiCom*, 2005.

[27] R. Ramanathan. Challenges: A radically new architecture for next generation mobile ad-hoc networks. In *Proceedings of ACM MobiCom*, 2005.

[28] A. Raniwala and T. cker Chiueh. Architecture and algorithms for an IEEE 802.11-based multi-channel wireless mesh network. In *Proceedings of IEEE INFOCOM*, 2005.

[29] K. Sanzgiri, I. D. Chakeres, and E. M. Belding-Royer. Determining intra-flow contention along multihop paths in wireless networks. In *Proceedings of Broadnets Wireless Networking Symposium*, 2004.

[30] J. P. Sharad. Estimation of link interference in static multi-hop wireless networks.

[31] P. Sinha, T. Nandagopal, N. Venkitaraman, R. Sivakumar, and V. Bharghavan. WTC: a reliable transport protocol for wireless wide-area networks. In *Proceedings of ACM MobiCom*, 1999.

[32] R. Sivakumar, P. Sinha, and V. Bharghavan. CEDAR: a core-extraction distributed ad hoc routing algorithm. *IEEE Journal on Selected Areas in Communications, Special Issue on Ad hoc Networks*, 17(8), 1999.

[33] K. Sundaresan and R. Sivakumar. A unified MAC layer framework for ad-hoc networks with smart antennas. In *Proceedings of ACM MobiHoc*, 2004.

[34] C. Tschudin and E. Osipov. Estimating the ad hoc horizon for TCP over IEEE 802.11 networks. In *Proceedings of Med-Hoc-Net*, 2004.

[35] C. L. Van Jacobson and S. McCanne. tcpdump. `http://tcpdump.org`.

[36] K. Xu, M. Gerla, L. Qi, and Y. Shu. TCP unfairness in ad hoc wireless networks and a neighborhood RED solution. In *Proceedings of ACM MobiCom*, 2003.

[37] S. Xu and T. Saadawi. Does the IEEE 802.11 MAC protocol work well in multihop wireless ad hoc networks? *IEEE Communication Magazine*, 39(6):130–137, June 2001.