Comparison of Routing Metrics for Static Multi-Hop Wireless Networks

Richard Draves   Jitendra Padhye   Brian Zill

Microsoft Research
One Microsoft Way, Redmond, WA 98052.

{richdr, padhye, bzill}@microsoft.com

Technical Report
MSR-TR-2004-18

Routing protocols for wireless ad-hoc networks have traditionally focused on finding paths with the least number of intermediate hops. However, such paths can include slow or lossy links, leading to poor throughput. A routing algorithm can select better paths by explicitly taking into account the quality of the wireless links. Metrics to measure wireless link quality have been proposed, but their relative performance has not been studied. In this paper, we conduct a detailed, empirical evaluation of the performance of three link-quality metrics—ETX, per-hop RTT, and per-hop packet pair—and compare them against minimum hop count. We study these metrics using a DSR-based routing protocol running over a wireless testbed consisting of 23 nodes with 802.11a cards. We find that the ETX metric has the best performance when all nodes are stationary. We also find that the per-hop RTT and per-hop packet-pair metrics perform poorly due to self-interference. Interestingly, the hop-count metric outperforms all of the link-quality metrics in a scenario where the sender is mobile.

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

1 Introduction

Routing in ad-hoc wireless networks has been an active area of research for many years. Much of the original work in the area was motivated by mobile application environments, such as battlefield ad-hoc networks. The primary focus in such environments is to provide scalable routing in the presence of mobile nodes. Recently, interesting commercial applications of multi-hop wireless networks have emerged. One example of such applications is "community wireless networks" [7, 19, 29, 30]. In such networks, most of the nodes are either stationary or minimally mobile. The focus of routing algorithms in such networks is on improving the network capacity or the performance of individual transfers.

In static ad-hoc wireless networks, choosing paths that minimize hop count (the most commonly used metric in ad-hoc routing [18, 25–27]) can lead to poor performance [11] because such paths tend to include wireless links between distant nodes. These long wireless links can be slow or lossy, leading to poor throughput for flows that traverse them. A routing algorithm can select better paths by explicitly taking link quality into account.

Researchers have proposed some metrics to measure wireless link quality, but to our knowledge, the relative performance of these metrics for the purpose of routing in static ad-hoc wireless networks has not been investigated. In this paper, we report the results of an experimental study of three link-quality metrics, and compare them against minimum hop-count routing. The first metric is called "Expected Transmission Count" (ETX) [10], and is based on measuring the loss rate of broadcast packets between pairs of neighboring nodes. The second metric is called "Per-hop Round Trip Time"(RTT) [2]. This metric is based on measuring the round trip delay seen by unicast probes between neighboring nodes. The third metric is called "Per-hop Packet Pair Delay" (PktPair). This metric is based on measuring the delay between a pair of back-to-back probes to a neighboring node.

To compare the performance of these metrics, we incorporated them in an ad-hoc routing protocol, based on DSR [18]. The protocol runs on a 23-node testbed. The nodes are equipped with 802.11a cards, and run Windows XP. We experiment with various traffic scenarios such as long-lived TCP flows, multiple simultaneous data transfers and simulated web traffic. We also consider a scenario involving some mobility.

The main contributions of the paper are the following: (i) We describe a routing protocol, that incorporates the notion of link quality metrics, for routing in static ad-hoc wireless networks. We describe several implementation issues and optimizations. (ii) We present detailed experimental results to show that in scenarios with stationary nodes, the ETX metric out-performs hop-count although it uses longer paths. This is in contrast with the results in [10], in which the authors saw little or no gain from ETX in a DSR-based routing protocol. (iii) We show that the one-hop RTT and one-hop packet-pair metrics perform poorly, because their load-sensitivity leads to self-interference. Fourth, we also consider a scenario where we add a single mobile node to our 23-node static testbed. (iv) We show that in such a scenario minimum hop-count routing performs considerably better than link-quality routing because the metrics do not react sufficiently quickly.

The rest of the paper is organized as follows. In Section 2, we describe the three wireless link quality metrics that we consider in this paper. In Section 3, we describe our routing protocol. In Section 4, we describe the testbed. In Section 5, we present the main body of our results. In Section 6, we describe the scenario involving a mobile node. In Section 7, we provide an overview of related work. Section 8 concludes the paper.

2 Link Quality Metrics

We consider three wireless link quality metrics in this paper, which are described in detail below. We also support minimum hop-count routing by defining a "HOP" metric. Each of these metrics represents a different notion of what constitutes good link quality. In Section 7, we will discuss other link quality metrics that we have not included in this study.

In the following discussion, we will talk about links between wireless nodes. The process of link discovery (i.e. neighbor discovery), maintenance of this information (i.e. detecting when a link is broken) and the propagation of this information throughout the network is described in Section 3.

2.1 Hop Count (HOP)

This metric provides minimum hop-count routing. Link quality for this metric is a binary concept, either the link exists or it doesn't.

The primary advantage of this metric is its simplicity. Once the topology is known, it is easy to compute and minimize the hop count between a source and a destination. Moreover, computing the hop count requires no additional measurements, unlike the other metrics we will describe in this section.

The primary disadvantage of this metric is that it does not take packet loss or bandwidth into account. It has been shown [11] that a route that minimizes the hop count does not necessarily maximize the throughput of a flow. For example, a two-hop path over reliable or fast links can exhibit better performance than a one-hop path over a lossy or slow link. The HOP metric, however, will prefer the one-hop path.

2.2 Per-hop Round Trip Time (RTT)

This metric is based on measuring the round trip delay seen by unicast probes between neighboring nodes. Adya et al. [2] proposed this metric. To calculate RTT, a node sends a probe packet carrying a timestamp to each of its neighbors every 500 milliseconds. Each neighbor immediately responds to the probe with a probe acknowledgment echoing the timestamp. This enables the sending node to measure round trip time to each of its neighbors. The node keeps an exponentially weighted moving average of the RTT samples to each of its neighbors. The average computation is as follows:

$$\text{Average} = 0.1 \times \text{RTT Sample} + 0.9 \times \text{Average}$$

If a probe or a probe response packet is lost, the moving average is increased by 20% to reflect this loss. Similar penalty is taken if loss of a data packet is detected on the link. We also increase the moving average if we detect a loss of data packet. The routing algorithm selects the path with the least total sum of RTTs.

The RTT metric measures several different facets of link quality. First, if either the node or the neighbor is busy, the probe or the probe-ack packet will experience queuing delay, resulting in high RTT. Second, as shown in [2], if other nodes in the vicinity are busy, the probe or the probe-ack packet will experience delays due to channel contention, again resulting in high RTT. Third, if link between the nodes is lossy, the 802.11 ARQ mechanism may have to retransmit the probe or the probe-ack packet several times to get it delivered correctly. This also increases the RTT along that hop. Finally, if despite the ARQ mechanism, a probe or a probe-ack packet is lost, the node that sent the probe detects it, and increases the moving average as described earlier. In short, the RTT metric is designed to avoid highly loaded or lossy links.

We originally thought that it might be advantageous to use a metric that measured channel load and preferred those links with less load. However, as we discuss later in the paper, we observed that this leads to route instability due to a phenomenon we call self-interference. Reporting a good metric (i.e. lightly-loaded) for the link will result in that link becoming preferred, thereby increasing the traffic on the link. This increases the RTT on that link, eventually resulting in a bad metric being reported for the link. This phenomenon is made worse by the fact that due to limitations of our implementation, we are unable to insert the probe packets at the head of the queue maintained by the driver. This queuing delay significantly distorts the RTT value on that hop. The authors of [2] have also observed this phenomenon. This phenomenon, and associated route flapping has also been observed in wired networks. Some remedies have been suggested [3, 21]. We will discuss this further in Section 5.

This metric has several other disadvantages as well. First, there is the overhead of measuring the round trip time. We reduce this overhead by using small probe packets (137 bytes). Second, the metric doesn't explicitly take link data rate into account. We may be able to take impact of link data rate into account by using larger probe packets. However, larger probes would impose an even greater measurement overhead. Finally, this measurement technique requires that every pair of neighboring nodes probe each other. This implies that the technique might not scale to dense networks.

2.3 Per-hop Packet Pair Delay (PktPair)

This metric is based on measuring the delay between a pair of back-to-back probes to a neighboring node. It is designed to correct the problem of distortion of RTT measurement due to queuing delays. The packet-pair technique is well-known in the world of wired networks [20].

To calculate this metric, a node sends two probe packets back-to-back to each neighbor every 2 seconds. The first probe packet is small (137 bytes), and the next one is large (1000 bytes). The neighbor starts a timer upon receiving the first probe packet, and stops the timer upon receiving the second packet. It then sends the value of this timer, which is the delay between the receipt of the first and the second packet, back to the sending node. The sender maintains a exponentially weighted moving average of these delays for each of its neighbors, using the averaging equation described in the previous section. The objective of the routing algorithm is to minimize the sum of these delays.

Like the RTT metric, this metric also measures several facets of link quality. If, due to high loss rate, the second probe packet requires retransmissions by 802.11 ARQ, the delay measured by the neighbor will increase. If the link from the node to its neighbor has low bandwidth, the second packet will take more time to traverse the link, which will result in increased delay. If there is traffic in the vicinity of this hop, it will also result in increased delay, since the probe packets have to contend for the channel.

The primary advantage of this metric over RTT is that it isn't affected by queueing delays at the sending node, since both packets in a pair will be delayed equally. In addition, using a larger packet for the second probe makes the metric more sensitive to the link bandwidth than the RTT metric.

This metric has several disadvantages. First, it is subject to overheads even greater than those of the RTT metric, since two packets are sent to each neighbor, and the second packet is larger. Second, we discovered that the metric is not completely immune to the phenomenon of self-interference. To understand self-interference for packet-pair measurements, consider three wireless nodes A, B, and C forming a two-hop chain topology. Assume that A is sending data to C via B. If a queue builds up on the link from A to B, the PktPair measurements on that link won't be affected because both the probe packets would be delayed equally. Now consider the link from B to C. Node B can not simultaneously receive a packet from A and send a probe to C. This means that the probe packet is contending with the data packet for the wireless channel. This increases the metric from B to C, increasing the total metric along the path from A to C. This self-interference is less severe than that experienced by RTT. However, as we will see, it does affect PktPair performance adversely.

2.4 Expected Transmission Count (ETX)

This metric estimates the number of retransmissions needed to send unicast packets by measuring the loss rate of broadcast packets between pairs of neighboring nodes. De Couto et al. [10] proposed ETX. To compute ETX, each node broadcasts a probe packet every second. The probe contains the count of probes received from each neighboring node in the previous 10 seconds. Based on these probes, a node can calculate the loss rate of probes on the links to and from its neighbors. Note that the broadcast packets are not retransmitted by the 802.11 ARQ mechanism. This allows the node to estimate the number of times the 802.11 ARQ mechanism will retransmit a unicast packet.

To illustrate this, consider two nodes A and B. Assume that node A has received 8 probe packets from B in the previous 10 seconds, and in the last probe packet, B reported that it had received 9 probe packets from A in the previous 10 seconds. Thus, the loss rate of packets from A to B is 0.1, while the loss rate of packets from B to A is 0.2. A successful unicast data transfer in 802.11 involves sending the data packet and receiving a link-layer acknowledgment from the receiver. Thus, the probability that the data packet will be successfully transmitted from A to B in a single attempt is $(1-0.1)\times(1-0.2) = 0.72$. If either the data or the ack is lost, the 802.11 ARQ mechanism will retransmit the packet. If we assume that losses are independent, the expected number of retransmissions before the packet is successfully delivered is $1/0.72 = 1.39$. This is the value of the ETX metric for the link from A to B. The routing protocol finds a path that minimizes the sum of the expected number of retransmissions.

Note that A calculates a new ETX value for the link from A to B every time it receives a probe from B. In our implementation of the ETX metric, the node maintains an exponentially weighted moving average of ETX samples, using the same averaging equation as the RTT metric. Note that there is no question of taking 20% penalty for lost probe packets. We do, however, take the penalty if loss of a data packet is detected.

ETX has several advantages. Since each node broadcasts the probe packets instead of unicasting them, the probing overhead is substantially reduced. The metric suffers little from self-interference since we are not measuring delays.

The main disadvantage of this metric is that since broadcast probe packets are small, and are sent at the lowest possible data rate (6Mbps in case of 802.11a), they may not experience the same loss rate as data packets sent at higher rates do. Moreover, the metric does not directly account for link load or data rate. A heavily loaded link may have very low loss rate, and two links with different data rates may have the same loss rate.

3 Ad-hoc Routing Architecture

We implement ad-hoc routing and link-quality measurement in a module that we call the Mesh Connectivity Layer (MCL). Architecturally, MCL is a loadable Windows driver. It implements a virtual network adapter, so that to the rest of the system the ad-hoc network appears as an additional (virtual) network link. MCL routes using a modified version of DSR [1] that we call Link-Quality Source Routing (LQSR). We have modified DSR extensively to improve its behavior, most significantly to support link-quality metrics. In this section, we review our architecture and implementation to provide background for understanding the performance results.

3.1 Ad-Hoc Routing at Layer 2.5

The MCL driver implements an interposition layer between layer 2 (the link layer) and layer 3 (the network layer). To higher-layer software, MCL appears to be just another ethernet link, albeit a virtual link. To lower-layer software, MCL appears to be just another protocol running over the physical link. See Figure 1 for a diagram.

IPv4	IPv6	IPX, etc.
Mesh Connectivity Layer (with LQSR)		
Ethernet	802.11	802.16, etc.

Figure 1: The ad-hoc routing implementation multiplexes multiple physical links into a single virtual link, over which we run unmodified network protocols and applications.

This design has several significant advantages. First, higher-layer software runs unmodified over the ad-hoc network. In our testbed, we run both IPv4 and IPv6 over the ad-hoc network. No modifications to either network stack were required. Network layer functionality, for example ARP, DHCP, and Neighbor Discovery, just works. Second, the ad-hoc routing runs over heterogeneous link layers. Our current implementation supports ethernet-like physical link layers (eg 802.11 and 802.3) but the architecture accommodates link layers with arbitrary addressing and framing conventions. The virtual MCL network adapter can multiplex several physical network adapters, so the ad-hoc network can extend across heterogeneous physical links. Third, the design can support any ad-hoc routing protocol, including DSR [18], AODV [27], TORA [25], etc.

In the simple configuration shown in Figure 1, the MCL driver binds to all the physical adapters and IP binds only to the MCL virtual adapter. This avoids multi-homing at the IP layer; for example, the mesh node then has a single IPv4 address. However other configurations are certainly possible. In our testbed deployment, the nodes have both an 802.11 adapter for the ad-hoc network and an ethernet adapter for management and diagnosis. We configure MCL to bind only to the 802.11 adapter. The IP stack binds to both MCL and the ethernet adapter. Hence the mesh nodes are multi-homed at the IP layer, so they have both a mesh IP address and a management IP address. We prevent MCL from binding to the management ethernet adapter, so the ad-hoc routing does not discover the ethernet as a high-quality single-hop link between all mesh nodes.

The virtual MCL network adapter appears to higher-layer software as an ethernet link. The MCL adapter has its own 48-bit virtual ethernet address, distinct from the layer-2 addresses of the underlying physical adapters. The mesh network functions just like an ethernet, except that it has a smaller MTU. To allow room for the LQSR headers, it exposes a 1280-byte MTU instead of the normal 1500-byte ethernet MTU. Our Windows 802.11 drivers do not support the maximum 2346-byte 802.11 frame size.

3.2 Modifications to DSR

Our MCL driver implements a version of Dynamic Source Routing (DSR) that we call Link-Quality Source Routing (LQSR). LQSR implements all the basic DSR functionality, including Route Discovery (Route Request and Route Reply messages) and Route Maintenance (Route Error messages). LQSR uses a link cache instead of a route cache, so fundamentally it is a link-state routing protocol. The primary changes in LQSR versus DSR relate to its implementation at layer 2.5 instead of layer 3 and its support for link-quality metrics.

3.3 Modifications to support Layer 2.5 routing

Due to the layer 2.5 architecture, LQSR uses 48-bit virtual ethernet addresses. All LQSR headers, including Source Route, Route Request, Route Reply, and Route Error, use 48-bit virtual addresses instead of 32-bit IP addresses. To support multiple physical network interfaces per node, the 48-bit addresses are augmented with 8-bit interface indices. Each node locally assigns interface indices to its physical network adapters, starting at one. Two nodes may be connected by multiple links, for example if the nodes have multiple radios. To uniquely specify a link, LQSR uses the source virtual address, the outgoing interface index, the incoming interface index, and the destination virtual address.

3.3.1 Modifications to support Link-quality metrics

We have modified DSR in several ways to support routing according to link-quality metrics. These include modifications to Route Discovery and Route Maintenance plus new mechanisms for Metric Maintenance. Our design does not assume that the link-quality metric is symmetric.

First, LQSR Route Discovery supports link metrics. When a node receives a Route Request and appends its own address to the route in the Route Request, it also appends the metric for the link over which the packet arrived. When a node sends a Route Reply, the reply carries back the complete list of link metrics for the route.

Once Route Discovery populates a node's link cache, the cached link metrics must be kept reasonably up-to-date for the node's routing to remain accurate. In Section 5.2 we show that link metrics do vary considerably, even when nodes are not mobile. LQSR tackles this with two separate Metric Maintenance mechanisms. The first mechanism maintains the metrics for links that the node is using actively to route its packets, and second mechanism maintains the metrics of other links. In combination, the two Metric Maintenance mechanisms ensure that a node uses good routes in the face of changing metrics.

LQSR uses a reactive mechanism to maintain the metrics for the links which it is actively using. When a node sends a source-routed packet, each intermediate node updates the source route with the current metric for the next (outgoing) link. This carries up-to-date link metrics forward with the data. To get the link metrics back to the source of the packet flow (where they are needed for the routing computation), we have the recipient of a source-routed data packet send a gratuitous Route Reply back to the source, conveying the up-to-date link metrics from the arriving Source Route. This gratuitous Route Reply is delayed up to one second waiting for a piggy-backing opportunity, and while delayed, subsequent gratuitous Route Replies squash (replace) earlier delayed Route Replies. This design keeps the overhead low while keeping the source of a packet flow informed about changes in link metrics along the route.

LQSR uses a proactive background mechanism to maintain the metrics for all links. Occasionally each LQSR node send a Link Info message. The Link Info carries current metrics for each link from the originating node, including broken links with an infinite metric. The Link Info is piggy-backed on a Route Request, so it floods throughout the neighborhood of the node. LQSR attempts to piggy-back Link Info messages on all Route Requests, if there is room in the packet. If a node has not sent a Link Info in the last 10 seconds, then it generates a dummy Route Request for the purpose of carrying a Link Info message.

The link metric support also affects Route Maintenance. When Route Maintenance notices that a link is not functional (because a requested Ack has not been received), it penalizes the link's metric and sends a Route Error. The Route Error carries the link's updated metric back to the source of the packet.

3.3.2 Other modifications

Our LQSR implementation includes the usual DSR conceptual data structures. These include a Send Buffer, for buffering packets while performing Route Discovery; a Maintenance Buffer, for buffering packets while performing Route Maintenance; and a Route Request table, for suppressing duplicate Route Requests. Instead of a Route Cache, we use a Link Cache and run Dijkstra's algorithm to calculate routes.

The LQSR implementation of Route Discovery omits some optimizations that are not worthwhile in our environment. In practice, Route Discovery is almost never required in our testbed so we have not optimized it. In particular, LQSR nodes do not reply to Route Requests from their link cache. Only the target of a Route Request sends a Route Reply. Furthermore, nodes do not send Route Requests with a hop limit to restrict their propagation. Route Requests always flood throughout the ad-hoc network. Nodes do cache information from overheard Route Requests.

The Windows 802.11 drivers do not support promiscuous mode and they do not indicate whether a packet was successfully transmitted. Hence our implementation of Route Maintenance uses explicit acknowledgments instead of passive acknowledgments or link-layer acknowledgments. Every source-routed packet carries an Ack Request option. A node expects an Ack from the next hop within 500ms. The Ack options are delayed briefly (up to 80ms) so that they may be piggy-backed on other packets flowing in the reverse direction. Also later Acks squash (replace) earlier Acks that are waiting for transmission. As a result of these techniques, the acknowledgment mechanism does not add significant byte or packet overhead.

The LQSR implementation of Route Maintenance also omits some optimizations. LQSR nodes do not implement "Automatic Route Shortening," which would allow nodes to send a gratuitous Route Reply when they overhear a source-routed packet before it is routed to them. Route shortening is not possible because of the lack of promiscuous mode. LQSR nodes do not implement "Increased Spreading of Route Error Messages," which would piggy-back the last-received Route Error on the next Route Request. This is not important because LQSR will not reply to a Route Request from (possibly stale) cached data. When LQSR Route Maintenance detects a broken link, it does not remove from the transmit queue other packets waiting to be sent over the broken link. Unfortunately, Windows drivers do not provide access to the transmit queue. However, LQSR nodes do learn from Route Error messages that they forward.

LQSR supports a form of DSR's "Packet Salvaging" or retransmission. Salvaging allows a node to try a different route when it is forwarding a source-routed packet and discovers that the next hop is not reachable. The acknowledgment mechanism does not allow every packet to be salvaged because it is primarily designed to detect when links fail. When sending a packet over a link, if the link has recently (within 250ms) been confirmed to be functional, we request an Ack as usual but we do not buffer the packet for possible salvaging. This design allows for salvaging the first packets in a new connection and salvaging infrequent connection-less communication, but relies on transport-layer retransmission for active connections. In our experience, packets traversing "cold" routes are more vulnerable to loss from stale routes and benefit from the retransmission provided by salvaging.

Our LQSR implementation does not assume that links are symmetric. For example, it sends Route Replies using an independently-discovered source route instead of blindly reversing the route in the Reply. Similarly, an Ack option may take a different (multi-hop) path back instead of reversing the one-hop path by which the Ack Request option arrived. Hence LQSR does not need the DSR Blacklist mechanism for detecting and avoiding asymmetric links.

We have not yet implemented the DSR "Flow State" optimization, which uses soft-state to replace a full source route with a small flow identifier. We intend to implement it in the future. Our Link Cache implementation does not use the Link-MaxLife algorithm [1] to timeout links. We found that Link-MaxLife produced inordinate churn in the link cache. Instead, we use an infinite metric value to denote broken links in the cache. We garbage-collect dead links in the cache after a day.

4 Testbed

The experimental data reported in this paper are the results of measurements we have taken on a 23-node wireless testbed. Our testbed is located on one floor of a fairly typical office building, with the nodes placed in offices, conference rooms and labs. Unlike wireless-friendly cubicle environments, our building has rooms with floor-to-ceiling walls and solid wood doors. With the exception of one additional laptop used in the mobility experiments, the nodes are located in fixed locations and did not move during testing. The node density was deliberately kept high enough to enable a wide variety of multi-hop path choices. See Figure 2.

The nodes are primarily laptop PCs with Intel Pentium II processors with clock rates from 233 to 300 MHz, but also included a couple slightly faster laptops as well as two desktop machines. All of the nodes run Microsoft Windows XP. The TCP stack included with XP supports the SACK option by default, and we left it enabled for all of our TCP experiments. All of our experiments were conducted over IPv4 using statically assigned addresses.

Each node has an 802.11a PCCARD radio. We used the default configuration for the radios, except for configuring adhoc mode and channel 36 (5.18 GHz). In particular, the cards all performed auto-speed selection. There are no other 802.11a users in our building.

We use four different types of cards in our testbed: 11 Proxim ORiNOCO ComboCard Gold, 7 NetGear WAG 511, 4 NetGear WAB 501, and 1 Proxim Harmony. While we performed no formal testing of radio ranges, we observed that some cards exhibited noticeably better range than others. The Proxim ORiNOCOs had the worst range of the cards we used in the testbed. The NetGear WAG 511s and WAB 501s exhibited range comparable to each other, and somewhere between the two Proxim cards. The Proxim Harmony had the best range of the cards we tried.

5 Results

In this section, we describe the results of our experiments. The section is organized as follows. First, we present measurements that characterize our testbed. These include a study of the overhead imposed by our routing software, and a characterization of the variability of wireless links in our testbed. Then, we present experiments that compare the four routing metrics under various type of traffic.

5.1 LQSR Overhead

Like any routing protocol, LQSR incurs a certain amount of overhead. First, it adds additional traffic in the form of routing updates, probes, etc. Second, it has the overhead of carrying the source route and other fields in each packet. Third, all nodes along the path of a data flow sign each packet using HMAC-SHA1 and regenerate the hash to reflect the changes in the LQSR headers when forwarding the packet. Also, the end nodes encrypt or decrypt the payload data using AES-128 [1]. The cryptographic overhead can be significant given the slow CPU speeds of the nodes in our testbed.

The following experiment measures the overhead of LQSR. We used four laptops named A, B, C and D. These machines were similar to those used in the testbed. Each machine was equipped with a Proxim ORiNOCO card. The machines were placed in close proximity of each other. All the links between the machines were operating at the maximum data rate (nominally 54Mbps).

[1]Our network administrators insisted upon better protection than WEP before they would allow us to connect our testbed to the campus network.

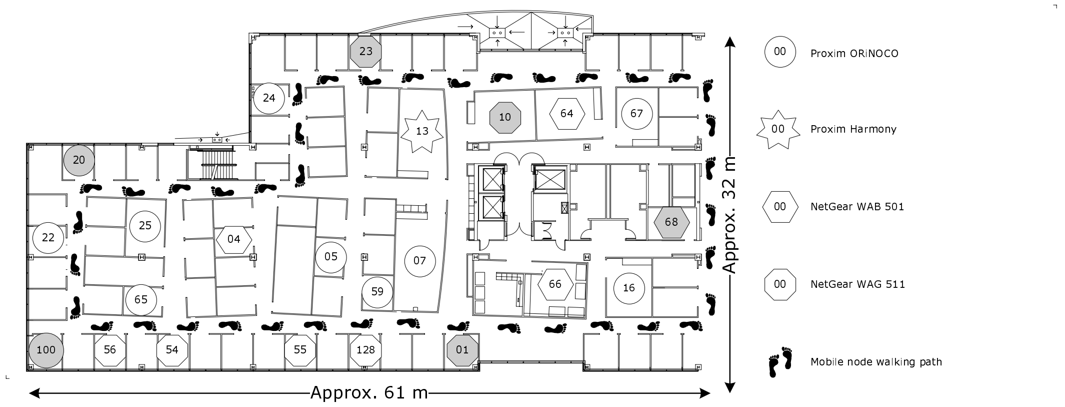

Figure 2: Our testbed consists of 23 nodes placed in fixed locations inside an office building. Four different models of 802.11a wireless cards were used. The six shaded nodes were used as endpoints for a subset of the experiments (see section 5.3). The mobile node walking path shows the route taken during the mobility experiments (see section 6).

To establish a baseline for our measurements, we set up static IP routes between these machines to form a chain topology. In other words, a packet from A to D was forwarded via B and C, and a packet from D to A was forwarded via C and B. We measured the throughput of long-lived TCP flows from A to B (a 1-hop path), from A to C (a 2-hop path) and from A to D (a 3-hop path). At any time, only one TCP flow was active. These throughputs form our baseline.

Next, we deleted the static IP routes and started LQSR on each machine. LQSR allows the user to set up static routes that override the routes discovered by the routing protocol. We set up static routes to form the same chain topology described earlier. Note that LQSR continues to send its normal control packets and headers, but routes discovered through this process are ignored in favor of the static routes supplied by the user. We once again measured the throughput of long-lived TCP flows on 1, 2 and 3 hop paths. Finally, we turned off all cryptographic functionality and measured the throughput over 1, 2 and 3 hop paths again.

The results of these experiments are shown in Figure 3. Each bar represents the average of 5 TCP connections. The variation between runs is negligible. The first thing to note is that, as one would expect, the throughput falls linearly with the number of hops due to inter-hop interference. LQSR's overhead is most evident on 1-hop paths. The throughput reduction due to LQSR, compared to static IP routing is over 38%. However, when cryptographic functionality is turned off, the throughput reduction is only 13%. Thus, we can conclude that the LQSR overhead is largely due to cryptography, which implies that the CPU is the bottleneck. The amount of overhead imposed by LQSR decreases as the path length increases. This is because channel contention between successive hops is the dominant cause of throughput reduction. At these reduced data rates, the CPU can easily handle the cryptographic overhead. The remaining overhead is due to the additional network traffic and headers carried in each packet. Note that the amount of control traffic varies depending on the number of neighbors that a node has. This is especially true for the RTT and PktPair metrics. We believe that this variation is not significant.

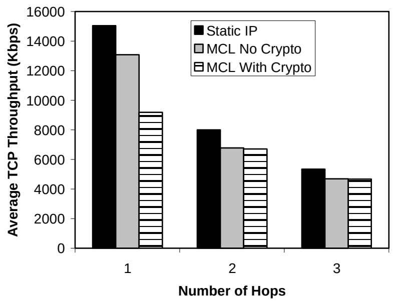

Figure 3: MCL does not significantly impact performance on multi-hop paths. On single-hop paths the cryptography overhead is significant.

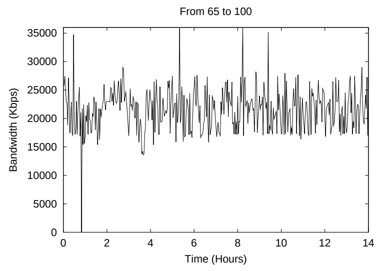

Figure 4: The bandwidth of the link from 65 to 100 is roughly stable over time.

5.2 Link Variability in the Testbed

In our testbed, we allow the radios to dynamically select their own data rates. Thus, different links in our testbed have different bandwidths. To characterize this variability in link quality, we conducted the following experiment. Recall the PktPair metric collects a sample of the amount of time required to transmit a probe packet every 2 seconds on each link. We modified the implementation of the PktPair metric to keep track of the minimum sample out of every successive 50 samples (i.e minimum of samples 1-50, then minimum of samples 51-100 etc.). We divide the size of the second packet by this minimum. The resulting number is an indication of the bandwidth

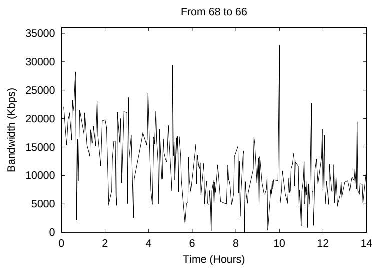

Figure 5: The bandwidth of the link between 68 and 66 appears to vary over time.

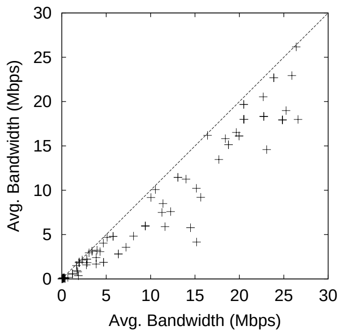

Figure 6: Each point corresponds to the two links between a pair of nodes. The X co-ordinate represents the link with the higher bandwidth. There are several node pairs with asymmetric bandwidths.

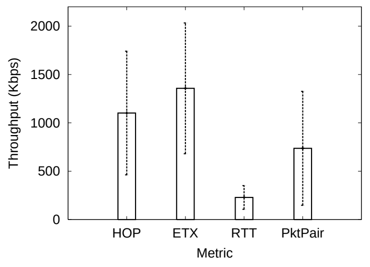

Figure 7: All pairs: Median throughput of three minute TCP transfers between every ordered pair of nodes (Total $23 \times 22 = 506$). Throughput is highest under ETX, lowest under RTT.

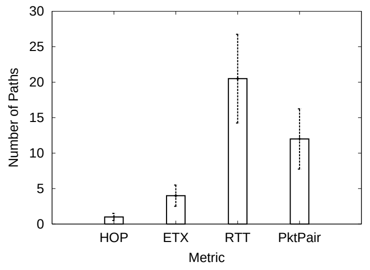

Figure 8: All Pairs: Median of number of paths taken by a TCP transfer. The impact of self-interference on RTT and PktPair metrics is evident.

of that link during that 100 second period. In controlled experiments, we verified that this approach approximates the raw link bandwidth. We gathered these samples from all links for a period of 14 hours. Thus, for each link there were a total of $14 \times 60 \times 60 \div 100 = 504$ bandwidth samples. There was no traffic on the testbed during this time. We discard any intervals in which the calculated bandwidth is more than 36Mbps, which is the highest data rate that we actually see in the testbed. This resulted in removal of 3.83% of all bandwidth samples. Still, the resulting number is not an exact measure of the available bandwidth, since it is difficult to correctly account for all link-layer overhead. However, we believe that the number is a good (but rough) indication of the link bandwidth during the 100 second period.

Of the $23 \times 22 = 506$ total possible links, only 183 links had non-zero average bandwidth, where the average was computed across all samples gathered over 14 hours. We found that bandwidths of certain links varied significantly over time, while other links it was relatively stable. Examples of two such links appear in Figures 4 and 5. In the first case, we see that the bandwidth is relatively stable over the duration of the experient. In the second case, however, the bandwidth is much more variable. Since the quality of links in our testbed varies over time, we are careful to repeat our experiments at different times.

In Figure 6 we compare the bandwidth on the forward and reverse direction of a link. To do this, we consider all possible unordered node pairs. The number of such pairs is $23 \times 22 \div 2 = 253$. Each pair corresponds to two directional links. Each of these two links has its own average bandwidth. Thus, each pair has two bandwidths associated with it. Out of the 253 possible node pairs, 90 node pairs had non-zero average bandwidth in both forward and reverse directions. In Figure 6, we plot a point for each such pair. The X-coordinate of the pair represents the link with the larger bandwidth. The existence of several points below the diagonal line implies that there are several pairs for which the forward and the reverse bandwidths differ significantly. In fact, in 47 node pairs, the forward and the reverse bandwidths differ by more than 25%.

5.3 Impact of Routing Metrics on Long Lived TCP Flows

Having characterized the overhead of our routing software, and the quality of links in our testbed, we can now discuss how various routing metrics perform in our testbed. We begin by discussing the impact of routing metrics on the performance of long-lived TCP connections. In today's Internet, TCP carries most of the traffic, and most of the bytes are carried as part of long-lived TCP flows [16]. It is reasonable to expect that similar types of traffic will be present on community networks, such as [7, 30]. Therefore, it is important to examine the impact of routing metrics on the performance of long-lived TCP flows.

We start the performance comparison with a simple experiment. We carried out a TCP transfer between each unique sender-destination pair. There are 23 nodes in our testbed, so a total of $23 \times 22 = 506$ TCP transfers were carried out. Each

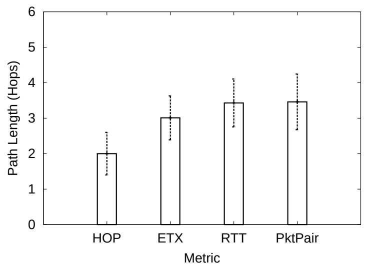

Figure 9: All pairs: Median path length of a TCP transfer. The HOP metric uses the shortest paths.

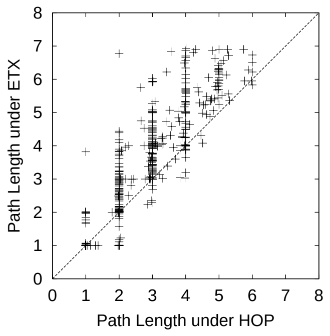

Figure 10: All Pairs: Comparison of HOP and ETX path lengths. Each point represents a connection. ETX consistently uses longer paths than HOP.

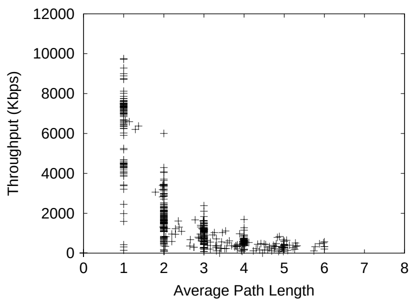

Figure 11: All Pairs: Throughput as a function of path length under HOP. The metric does a poor job of selecting multi-hop paths.

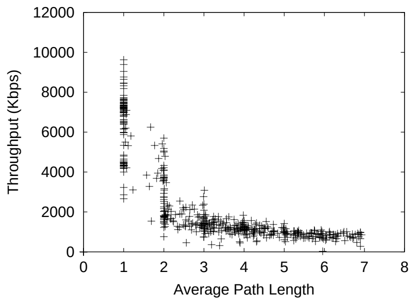

Figure 12: All Pairs: Throughput as a function of path length under ETX. The metric does a better job of selecting multi-hop paths.

TCP transfer lasted for 3 minutes, and transferred as much data as it could. On the best one-hop path in our testbed a 3 minute connection will transfer over 125MB of data. Such large TCP transfers ensure repeatability of results. We had previously determined empirically that TCP connections of 1 minute duration were of sufficient length to overcome startup effects and give reproducible results. To be conservative, we used 3 minute transfers in these experiments. [2] Only one TCP transfer was active at any time. The total time required for the experiment was just over 25 hours. We repeated the experiment for each metric.

In Figure 7 we show the median throughput of the 506 TCP transfers for each metric. We choose median to represent the data instead of the mean because the distribution (which includes transfers of varying path lengths) is quite skewed. The height error bars represent the *semi-inter quartile range* (SIQR), which is defined as half the difference between 25th and 75th percentile of the data. SIQR is the recommended measure of dispersion when the central tendency of the data is represented by the median [17]. Since each connection is run between a different pair of nodes the relatively large error bars indicate that we observe a wide range of throughputs across all the pairs. The median throughput using the HOP metric is 1100Kbps, while the median throughput using the ETX metric is 1357Kbps. This represents an improvement of 23.1%.

In contrast, De Couto et al. [10] observed almost no improvement for ETX in their DSR experiments. There are several possible explanations for this. First, they used UDP instead of TCP. A bad path will have more impact on the throughput of a TCP connection (due to window backoffs, timeouts etc.) than on the throughput of a UDP connection. Hence, TCP amplifies the negative impact of poor route selection. Second, in their testbed the radios were set to their lowest sending rate of 1Mbps, whereas we allow the radios to set transmit rates automatically (auto-rate). We believe links with lower loss rates also tend to have higher data rates, further amplifying ETX's improvement. Third, our testbed has 6-7 hop diameter whereas their testbed has a 5-hop diameter [9]. As we discuss below, ETX's improvement over HOP is more pronounced at longer path lengths.

The RTT metric gives the worst performance among the four metrics. This is due to the phenomenon of self-interference that we previously noted in Section 2.2. The phenomenon manifests itself in the number of paths taken by the connection. At the beginning the connection uses a certain path. However, due to self-interference, the metric on this path soon rises. The connection then chooses another path. This is illustrated in Figure 8. The graph shows the median number of paths taken by a connection. The RTT metric uses far more paths per connection than other metrics. The HOP metric uses the least number of paths per connection - the median is just 1.

The PktPair metric performs better than RTT, but worse than both HOP and ETX. This is again due to the phenomenon of self-interference. While the RTT metric suffers from self-interference on all hops along the path, the PktPair metric elim-

[2] We chose to fix the transfer duration, instead of the amount of data transferred for practical reasons. We shared our testbed with other researchers, and it was important that the running time of each experiment be predictable for scheduling purposes.

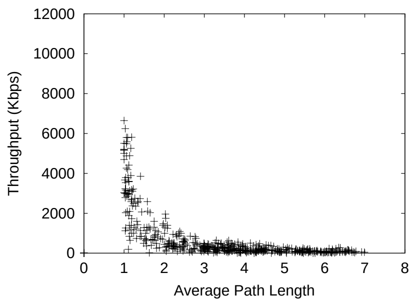

Figure 13: All Pairs: Throughput as a function of path length under RTT. The metric does a poor job of selecting even one hop paths.

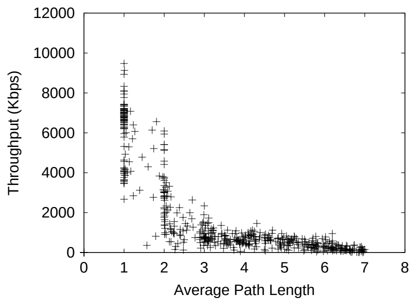

Figure 14: All Pairs: Throughput as a function of path length under PktPair. The metric finds good one-hop paths, but poor multi-hop paths.

inates the self-interference problem on the first hop. The impact of this can be seen in the median number of paths (12) tried by a connection using the PktPair metric. This number is lower than median using RTT (20.5), but substantially higher than HOP and ETX.

Note that the ETX metric also uses several paths per connection: the median is 4. This is because for a given node pair, multiple paths that are essentially equivalent can exist between them. There are several such node pairs in our testbed. Small fluctuations in the metric values of equivalent paths can make ETX choose one path over another. We plan to investigate route damping strategies to alleviate this problem.

The self-interference, and consequent route flapping experienced the RTT metric has also been observed in wired networks [3, 21]. In [21], the authors propose to solve the problem by converting the RTT to utilization, and normalizing the resulting value for use as a route metric. In [3], the authors propose to use hysteresis to alleviate route flapping. We are currently investigating these techniques further. Our initial results show that hysteresis may reduce the severity of the problem, but not significantly so.

5.4 Impact of Path Length

The HOP metric produces significantly shorter paths than the three other metrics. This is illustrated in Figure 9. The bar chart shows the median across all 506 TCP transfers of the average path length of each transfer. To calculate the average path length of a TCP transfer, we keep track of the paths taken by all the data-carrying packets in the transfer. We calculate the average path length by weighting the length of each unique path by the number of packets that took that path. The error bars represent SIQR. The HOP metric has the shortest median path length (2), followed by ETX (3.01), RTT (3.43) and PktPair (3.58).

We now look at ETX and HOP path lengths in more detail. In Figure 10, we plot the average path length of each TCP transfer using HOP versus the average path length using ETX. Again, the ETX metric produces significantly longer paths than the HOP metric. The testbed diameter is 7 hops using ETX and 6 hops using HOP.

We also examined the impact of average path length on TCP throughput. In Figure 12 we plot the throughput of a TCP connection against its path length using ETX while in Figure 11 we plot the equivalent data for HOP. First, note that as one would expect, longer paths produce lower throughputs because channel contention keeps more than one link from being active. Second, note that ETX's higher median throughput derives more from avoiding lower throughputs than from achieving higher throughputs. Third, ETX does especially well at longer path lengths. The ETX plot is flat from around 5 through 7 hops, possibly indicating that links at opposite ends of the testbed do not interfere. Fourth, ETX avoids poor one-hops paths whereas HOP blithely uses them.

We now look at the performance of RTT and PktPair in more detail. In Figure 13 we plot TCP throughput versus average path length for RTT while in Figure 14 we plot the data for PktPair. RTT's self-interference is clearly evident in the low throughputs and high number of connections with average path lengths between 1 and 2 hops. With RTT, even 1-hop paths are not stable. In contrast, with PktPair the 1-hop paths look good (equivalent to ETX in Figure 12) but self-interference is evident starting at 2 hops.

5.5 Variability of TCP Throughput using HOP and ETX

To measure the impact of routing metrics on the variability of TCP throughput, we carry out the following experiment. We select 6 nodes on the periphery of the testbed, as shown in Figure 2. Each of the 6 nodes then carried out a 3-minute TCP transfer to the remaining 5 nodes. The TCP transfers were set sufficiently apart to ensure that no more than one TCP transfer will be active at any time. There is no other traffic in the testbed. We repeated this experiment 10 times. Thus, there were a total of $6 \times 5 \times 10 = 300$ TCP transfers. Since each transfer takes 3 minutes, the experiment lasts for over 15 hours.

In Figure 15 we show the median throughput of the 300 TCP transfers using each metric. As before, the error bars represent SIQR. Once again, we see that the RTT metric is the worst performer, and the ETX metric outperforms the other three metrics by a wide margin.

The median throughput using the ETX metric is 1133Kbps, while the median throughput using the HOP metric is 807.5. This represents a gain of 40.3%. This gain is higher than the 23.15% obtained in the previous experiment because these ma-

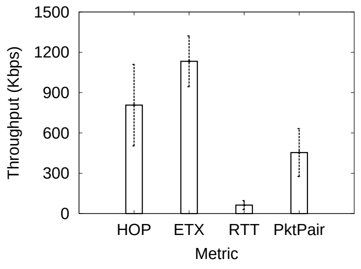

Figure 15: 30 Pairs: Median throughput with different metrics. The difference between HOP and ETX is higher than it is for All Pairs.

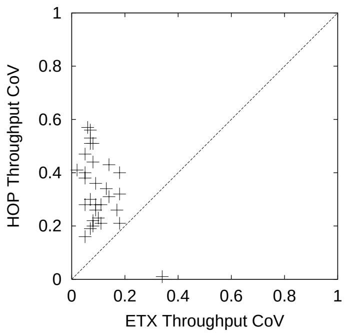

Figure 16: 30 Pairs: Each point is a CoV of throughputs of 10 TCP transfers between a given source-destination pairs using ETX and HOP. The CoVs are much lower under ETX, indicating that ETX chooses stable links.

chines are on the periphery of the network, and thus, the paths between them tend to be longer. As we have noted in Section 5.4, the ETX metric tends to perform better than HOP on longer paths. The higher median path lengths substantially degrades the performance of RTT and PktPair, compared to their performance shown in Figure 7.

The HOP metric selects the shortest path between a pair of nodes. If multiple shortest paths are available, the metric simply chooses the first one it finds. This introduces a certain amount of randomness in the performance of the HOP metric. If multiple TCP transfers are carried out between a given pair of nodes, the HOP metric may select different paths for each transfer. The ETX metric, on the other hand, selects "good" links. This means that it tends to choose the same path between a pair of nodes, as long as the link qualities do not change drastically. Thus, if several TCP transfers are carried out between the same pair of nodes at different times, they should yield similar throughput using ETX, while the throughput under HOP will be more variable. This fact is illustrated in Figure 16.

The figure uses coefficient of variation (CoV) as a measure of variability. CoV is defined as standard deviation divided by mean. There is one point in the figure for each of the 30 source-destination pairs. The X-coordinate represents CoV of the throughput of 10 TCP transfers conducted between a given source-destination pair, and the Y co-ordinate represents the CoV using HOP. The CoV values are significantly lower

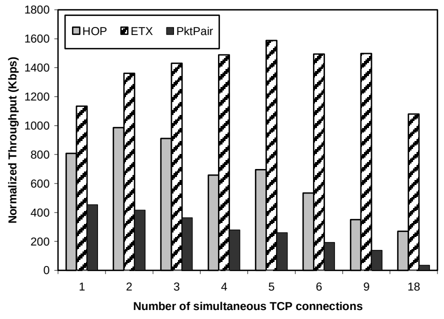

Figure 17: Throughputs with multiple simultaneous TCP connections.

with ETX. Note that a single point lies well below the diagonal line, indicating that HOP provided more stable throughput than ETX. This point represents TCP transfers from node 23 to node 10. We are currently investigating these transfers further. It is interesting to note that for the reverse transfers, i.e. from node 10 to node 23, ETX provides lower CoV than than HOP.

5.6 Multiple Simultaneous TCP Transfers

In the experiments described in the previous section, only one TCP connection was active at any time. This is unlikely to be the case in a real network. In this section, we compare the performance of ETX, HOP and PktPair for multiple simultaneous TCP connections. We do not consider RTT since its performance is poor even with a single TCP connection.

We use the same set of 6 peripheral nodes shown in Figure 2. We establish 10 TCP connections between each distinct pair of nodes. Thus, there are a total of $6 \times 5 \times 10 = 300$ possible TCP connections. Each TCP connection lasts for 3 minutes. The order in which the connections are established is randomized. The wait time between the start of two successive connections determines the number of simultaneously active connections. For example, if the wait time between starting consecutive connections is 90 seconds, then two connections will be active simultaneously. We repeat the experiment for various numbers of simultaneously active connections.

For each experiment we calculate the median throughout of the 300 connections, and multiply it by the number of simultaneously active connections. We call this product the Multiplied Median Throughput (MMT). MMT should increase with the number of simultaneous connections, until the load becomes too high for the network to carry.

In Figure 17 we plot MMT against the number of simultaneous active connections. The figure shows that the performance of the PktPair metric gets significantly worse as the the number of simultaneous connections increase. This is because the self-interference problem gets worse with increasing load. In the case of ETX, the MMT increases to a peak at 5 simultaneous connections. The MMT growth is significantly less than linear because there is not much parallelism in our testbed (many links interfere with each other) and the increase that we are seeing is partly because a single TCP connection does not fully utilize the end-to-end path. We believe the MMT falls beyond

5 simultaneous connections due to several factors, including 802.11 MAC inefficiencies and instability in the ETX metric under very high load. The MMT using HOP deteriorates much faster than it does with ETX. As discussed in Section 5.8, at higher loads HOP performance drops because link-failure detection becomes less effective.

5.7 Web-like TCP Transfers

Web traffic constitutes a significant portion of the total Internet traffic today. It is reasonable to assume that web traffic will also be a significant portion of traffic in wireless meshes such as the MIT Roofnet [29]. The web traffic is characterized by the heavy-tailed distribution of flow sizes: most transfers are small, but there are some very large transfers [24]. Thus, it is important to examine the performance of web traffic under various routing metrics.

To conduct this experiment, we set up a web server on host 128. The six peripheral nodes served as web clients. The web traffic was generated using Surge [6]. The Surge software has two main parts, a file generator and a request generator. The file generator generate files of varying sizes that are placed on the web server. The Surge request generator models a web user that fetches these files. The file generator and the request generator offer a large number of parameters to customize file size distribution and user behaviors. We ran Surge with its default parameter settings, which have been derived through modeling of empirical data [6].

Each Surge client modeled a single user running HTTP 1.1 Each user session lasted for 40 minutes, divided in four slots of 10 minutes each. Each user fetched over 1300 files from the web server. The smallest file fetched was 77 bytes long, while the largest was 700KB. We chose to have only one client active at any time, to allow us to study the behavior of each client in detail. We measure the latency or each object: the amount of time elapsed between the request for an object, and the completion of its receipt. Note that this is not an accurate estimate of the delay a client will see, since we are ignoring rendering delays.

In Figure 18, we plot the median latencies experienced by each client. It can be seen that ETX reduces the latencies observed by clients that are further away from the web server. This is consistent with our earlier finding ETX tends to perform better than HOP on longer paths. For host 23, 100 and 20, the median latency under ETX is almost 20% lower than the median latency under HOP. These hosts are relatively further away from the webserver running on host 128. On the other hand, for host 1, the median latency under HOP is lower by 20%. Host 1 is just one hop away from the web server. These results are consistent with the results in Section 5.4: on longer paths, ETX performs better than HOP, but on one-hop paths, the HOP metric sometimes performs better.

To study whether the impact of ETX is limited to large transfers we studied the median response times for small objects, i.e. files that are less than 1KB in size and large objects, i.e., those over 8KB in size. These medians are shown in Figures 19 and 20, respectively. The benefit of ETX is indeed more evident in case of larger transfers. However, ETX also reduces the latency of small transfers by significant proportion. This is particularly interesting as the data sent from the server to client in such small transfers fits inside a single TCP packet. It is clear that even for such short transfers, the paths selected by ETX are better.

5.8 Discussion

We conclude from our results that load-sensitivity is the primary factor determining the three metrics' relative performance. Although it appears at first to be a desirable attribute, load-sensitivity makes a metric susceptible to self-interference and hence causes poor performance. The RTT metric is the most sensitive to load; it suffers from self-interference even on one-hop paths and has the worst performance. The PktPair metric is not affected by load generated by the probing node, but it is sensitive to other load on the channel. This causes self-interference on multi-hop paths and degrades performance. The ETX metric has the least sensitivity to load and it performs the best. Furthermore, our preliminary measurements of non-rate-limited UDP traffic indicate that extreme traffic loads can affect ETX accuracy and impair throughput. De Couto et al. [10] have also observed this effect of non-rate-limited traffic on ETX.

Our experience with HOP leads us to believe that its performance is very sensitive to competing factors controlling the presence of poor links in the link cache. Consider the evolution of a route during a long data transfer. When the transfer starts, the link cache contains a fairly complete topology, including many poor links. The shortest-path Dijkstra computation picks a route that probably includes poor (lossy or slow) links. Then as the data transfer proceeds, Route Maintenance detects link failures and sends Route Error messages, causing the failed link to be removed from link caches. Since poor links suffer link failures more frequently than good links, over time the route tends to improve. However this process can go too far: if too many links are removed, the route can get longer and longer until finally there is no route left and the node performs Route Discovery again. On the other hand, a background load of unrelated traffic in the network tends to repopulate the link cache with good and bad links, because of the caching of link existence from overheard or forwarded packets. The competition between these two factors, one removing links from the link cache and the other adding links, controls the quality of the HOP metric's routes. For example, originally LQSR sent Link Info messages when using HOP. When we changed that, to make LQSR with HOP behave more like DSR, we saw a significant improvement in median TCP throughput. This is because the background load of Link Info messages was repopulating the link caches too quickly, reducing the effectiveness of the Route Error messages.

Our study has several limitations that we would like to correct in future work. First, our data traffic is entirely artificial. We would prefer to measure the performance of real network traffic generated by real users. Second, we do not investigate packet loss and jitter with constant-bit-rate datagram traffic.

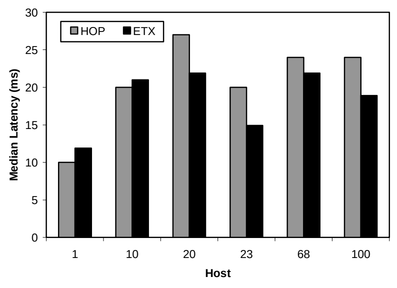

Figure 18: Median latency for all files fetched

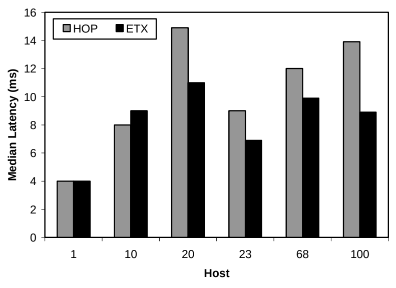

Figure 19: Median latency for files smaller than 1KB

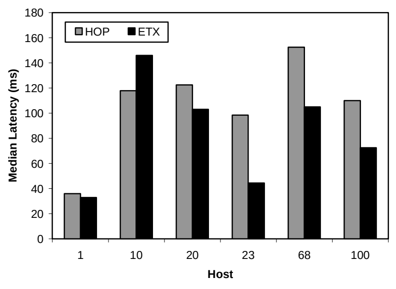

Figure 20: Median latency for files larger than 8KB

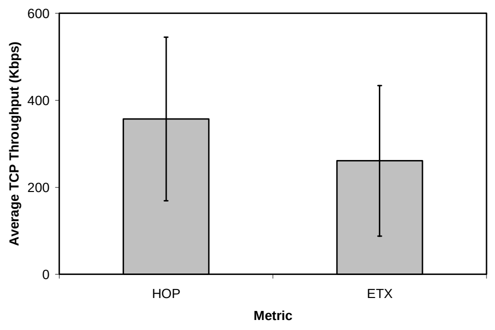

Figure 21: Average Throughput of 45 1-minute TCP transfers with mobile sender using HOP and ETX.

This would be relevant to the performance of multimedia traffic. We would also like to investigate performance of other wireless link quality metrics such as signal strength.

6 A Mobile Scenario

In the traffic scenarios that we have considered so far, all the nodes have been stationary. In community networks like [7, 29, 30] most nodes are indeed likely to be stationary. However, in most other ad-hoc wireless networks, at least some of the nodes are mobile. Here, we consider a scenario that involves a single mobile node, and compare the performance of ETX and HOP metrics.

The layout of our testbed is shown in Figure 2. We set up a TCP receiver on node 100. We slowly and steadily walked around the periphery of the network with a Dell Latitude Laptop, equipped with a NetGear card. A process running on this laptop repeatedly established a TCP connection to the receiver running on node 100, and transferred as much data as it could in 1 minute. We did 15 such transfers in each walk-about. We did 3 such walk-abouts each for ETX and HOP. Thus, for each metric we did 45 TCP transfers.

The median throughput of these 45 transfers, along with SIQR is shown in Figure 21. We choose median over mean since the distribution of throughputs is highly skewed. The median throughput under HOP metric is 36% higher than the median throughput under the ETX metric. Note also that the SIQR for ETX is 173, which is comparable to the SIQR of 188 for HOP. Since the median throughput under ETX is lower, the higher SIQR indicates greater variability in throughput under ETX.

As the sender moves around the network, the ETX metric does not react sufficiently quickly to track the changes in link quality. As a result, the node tries to route its packets using stale, and sometimes incorrect information. The salvaging mechanisms built into LQSR do help to some extent, but not well enough to overcome the problem completely. Our results with PktPair (not included here) indicate that that this problem is not limited to just the ETX metric. Any approach that tries to measure link quality will need some time to come up with a stable measure of link quality. If during this time the mobile user moves sufficiently, the link quality measurements would not be correct. Note that we do have penalty mechanisms built into our link quality measurements. If a data packet is dropped on a link, we penalize the metric as described in Section 2. We are investigating the possibility that by penalizing the metric more aggressively on data packet drops we can improve the performance of ETX.

The HOP metric, on the other hand, faces no such problems. It uses new links as quickly as the node discovers them. The efficacy of various DSR mechanisms to improve performance in a mobile environment has been well documented [18]. The metric also removes from link cache any link on which a node suffers even one packet loss. This mechanism, which hurts the performance of HOP metric under heavy load, benefits it in the mobile scenario.

We stress that this experiment is not an attempt to draw general conclusions about the suitability of any metric for routing in mobile wireless networks. Indeed, such conclusions can not be drawn from results of a single experiment. This experiment only serves to underscore the fact that static and mobile wireless networks can present two very different sets of challenges, and solutions that work well in one setting are not guaranteed to work just as well in another.

7 Related Work

There is a large body literature comparing the performance of various ad-hoc routing protocols. Most of this work is simulation-based and the ad-hoc routing protocols studied all minimize hop-count. Furthermore, many of these studies focus on scenarios that involve significant node mobility. For example, Broch et al. [8] compared the performance of DSDV [26], TORA [25], DSR [18], and AODV [27] using ns-2 [13] simulations of mobile nodes.

The problem of devising a link-quality metric for ad-hoc networking with 802.11 in neighborhood and office environments has been studied previously. Most notably, De Couto et al. [10] propose ETX and compare its performance to HOP using DSDV and DSR with a small-datagram workload. Their study differs from ours in many aspects, but it is the most closely-related work of which we know. They conclude that ETX outperforms HOP with DSDV, but find little benefit with DSR. De Couto et al. only study the throughput of single, short (30 second) data transfers using small datagrams. Their experiments include no mobility. In contrast, we study TCP transfers. We examine the impact of multiple simultaneous data transfers. We study variable-length data transfers and in particular, look at web-like workloads where latency is more important than throughput. Finally, our work includes a scenario with some mobility. De Couto's implementation of DSR differs from ours in several ways, which may partly explain our different results. Their DSR implementation only discovers link metrics via Route Request / Reply; they do not have Metric Maintenance mechanisms. In their testbed (as in ours), the availability of multiple paths means after the initial route discovery the nodes rarely send Route Requests. Hence during their experiments, the sender effectively routes using a snapshot of the ETX metrics discovered at the start of the experiment. [9] Their implementation takes advantage of 802.11 link-layer acknowledgments for failure detection. This means their link-failure detection is not vulnerable to loss, or perceived loss due to delay. Their implementation does not support salvaging. They mitigate this in their experiments by sending five "priming" packets before starting each data transfer. Their implementation uses a "blacklist" mechanism to cope with asymmetric links. Finally, their implementation has no security support and does not use cryptography so it has much less CPU overhead.

Woo et al. [31] examines the interaction of link quality and ad-hoc routing for sensor networks. Their work assumes passive observation of packet reception probability instead of active probing. They compare various filters for smoothing the reception probability. Using this as a basis, they compare several routing protocols including shortest-path routing with thresholding to eliminate links with poor quality and ETX-based distance-vector routing. Their study uses rate-limited datagram traffic. They conclude that ETX-based routing is more robust.

Signal strength, or signal-to-noise ratio (SNR), has been used as a link quality metric in several routing schemes for mobile ad-hoc networks. In [15] the authors use an SNR threshold value to filter links discovered by DSR Route Discovery. Similar ideas have also been explored in [12, 23, 32]. The main problem with these schemes is that they may end up excluding links that are poor in quality, but are still necessary to maintain connectivity. Another approach is used in [14], where links are still classified as good and bad based on a threshold value, but a path is permitted to use poor-quality links to maintain connectivity. Punnoose et. al. [28] also use signal strength as a link quality metric. They convert the predicted signal strength into a link quality factor, which is used assign weights to the links. The basic DSR algorithm then chooses the route with overall best link quality, instead of just the shortest route. Zhao and Govindan [33]. studied packet delivery performance in sensor networks, and discovered that high signal strength implies low packet loss, but low signal strength does not imply high packet loss. We plan to study the SNR metric in our testbed as part of our future work. Our existing hardware and software setup does not provide adequate support to study this metric.

Awerbuch et. al. [5] study impact of automatic rate selection on performance of ad hoc networks. They propose a routing algorithm that selects a path with minimum transmission time. Their metric does not take packet loss into account. It is possible to combine this metric with the ETX metric, and study performance of the combined metric. This is also part of our future work.

An implementation of AODV that uses the link-filtering approach, based on measurement of loss rate of unicast probes, was demonstrated in a recent IETF meeting [4, 22]. We tested this implementation in our testbed, and found it to be too unstable to complete our tests. We hope to get an improved version of the code soon, after which we will be able to conduct additional tests. We should note however, that the rate of unicast packet loss is very low in our testbed. The 802.11 MAC includes an ARQ mechanism retransmits a unicast packet several times. Since we are using a static testbed, these retransmissions are usually sufficient to ultimately get the packet delivered on all except very poor links.

8 Conclusions

We have examined the performance of three candidate link-quality metrics for ad-hoc routing and compared them to minimum hop-count routing. Our results are based on several months of experiments using a 23-node static ad-hoc network in an office environment. The results show that with stationary nodes the ETX metric significantly outperforms hop-count. The RTT and PktPair metrics perform poorly because they are load-sensitive and hence suffer from self-interference. However, in a mobile scenario hop-count performs better because it reacts more quickly to fast topology change.

References

[1] The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR), Apr. 2003. Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr-09.txt.

[2] A. Adya, P. Bahl, J. Padhye, A. Wolman, and L. Zhou. A Multi-Radio Unification Protocol for IEEE 802.11 Wireless Networks. Technical Report MSR-TR-2003-44, Microsoft Research.

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

[4] AODV@IETF. http://moment.cs.ucsb.edu/aodv-ietf/.

[5] B. Awerbuch, D. Holmer, and H. Rubens. High throughput route selection in mult-rate ad hoc wireless networks. Technical re-

port, Johns Hopkins University, Computer Science Department, March 2003. Technical Report, version 2.

[6] P. Bardford and M. Crovella. Generating representative web workloads for network and server performance evaluation. In *ACM SIGMETRICS*, Nov. 1998.

[7] Bay area wireless users group. http://www.bawug.org/.

[8] J. Broch, D. Maltz, D. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. *MOBICOM*, Oct. 1998.

[9] D. De Couto. Personal communication, Nov. 2003.

[10] D. De Couto, D. Aguayo, J. Bicket, and R. Morris. High-throughput path metric for multi-hop wireless routing. In *ACM MOBICOM*, Sept. 2003.

[11] D. S. J. De Couto, D. Aguayo, B. A. Chambers, and R. Morris. Performance of multihop wireless networks: Shortest path is not enough. In *1st Workshop on Hot Topics in Networks*, Oct. 2002.

[12] R. Dube, C. Rais, K.-E. Wang, and S. Tripathi. Signal stability based adaptive routing (ssa) for ad-hoc mobile networks. *IEEE Personal Communications*, 1997.

[13] K. Fall and e. K. Varadhan. *ns* notes and documentation. *The VINT Project, UC Berkeley*, Nov. 1997.

[14] T. Goff, N. Abu-Aahazaleh, D. Phatak, and R. Kahvecioglu. Preemptive routing in ad hoc networks. In *MOBICOM*, 2001.

[15] Y.-C. Hu and D. B. Johnson. Design and demonstration of live audio and video over muulti-hop wireless networks. In *MILCOM 2002*, 2002.

[16] P. Huang and J. Heidemann. Capturing tcp burstiness for lightweight simulation. In *In Proceedings of the SCS Multiconference on Distributed Simulation*, Jan. 2001. Phoneix, Arizona, Society for Computer Simulation.

[17] R. Jain. *The Art of Computer Systems Performance Analysis*. John Wiley and Sons, Inc., 1991.

[18] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad-hoc wireless networks. In T. Imielinski and H. Korth, editors, *Mobile Computing*. Kluwer Academic Publishers, 1996.

[19] R. Karrer, A. Sabharwal, and E. Knightly. Enabling Large-scale Wireless Broadband: The Case for TAPs. In *HotNets*, 2003.

[20] S. Keshav. A Control-theoretic approach to flow control. In *ACM SIGCOMM*, Sept. 1991.

[21] A. Khanna and J. Zinky. The Revised ARPANET Routing Metric. In *Proc ACM SIGCOMM*, 1989.

[22] L. Krishnamurthy. Personal communication, Dec. 2003.

[23] H. Lundgren, E. Nordstrom, and C. Tschudin. Coping with communication gray zones in IEEE 802.11b based ad hoc networks. In *ACM workshop on wireless mobile multimedia*, 2002.

[24] K. Park, G. Kim, and M. Crovella. On the relationship between file sizes, transport protocols and self-similar network tarffic. In *Proceedings of ICNP'96*, 1996.

[25] V. D. Park and M. S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *INFOCOM*, 1997.

[26] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance vector routing (dsdv) for mobile computeres. In *Proc. of ACM SIGCOMM'94*, Sep. 1994.

[27] C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. In *Proc. of IEEE WMCSA'99*, Feb. 1999.

[28] R. Punnose, P. Nitkin, J. Borch, and D. Stancil. Optimizing wireless network protocols using real time predictive propagation modeling. In *RAWCON*, August 1999.

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

[30] Seattle wireless. http://www.seattlewireless.net/.

[31] A. Woo, T. Tong, and D. Culler. Taming the underlying challenges of reliable multihop routing in sensor networks. In *SENSYS*, Nov. 2003.

[32] K. wu Chin, J. Judge, A. Williams, and R. Kermode. Implemenation experience with manet routing protocols. *ACM Computer Communication Review*, 32(5), November 2002.

[33] J. Zhao and R. Govindan. Understanding packet delivery performance in dense wireless sensor networks. In *SENSYS*, Nov. 2003.