Sustaining Cooperation in Multi-Hop Wireless Networks

Ratul Mahajan Maya Rodrig David Wetherall John Zahorjan

University of Washington

Abstract – Multi-hop wireless networks are vulnerable to free-riders because they require nodes to forward packets for each other. Deployed routing protocols ignore this issue while proposed solutions incorporate complicated mechanisms with the intent of making free-riding impossible. We present Catch, a protocol that falls between these extremes. It achieves nearly the low mechanism requirements of the former while imposing nearly as effective barriers to free-riding as the latter. Catch is made possible by novel techniques based on anonymous messages. These techniques enable cooperative nodes to detect nearby free-riders and disconnect them from the rest of the network. Catch has low overhead and is broadly applicable across routing protocols and traffic workloads. We evaluate it on an 802.11 wireless testbed as well as through simulation.

1 Introduction

Selfish behavior is an important design consideration whenever parties with varied interests come together to achieve a common goal. Examples where individual behavior can be at odds with the system goal include free-riding in peer-to-peer file sharing networks [1, 36, 25, 32, 13, 41], cheating in online games [33, 6], ISP competition in Internet routing [38, 12], and network congestion control [16, 17, 37, 23, 4, 26, 28]. As has been observed in many of these systems, some parties will behave selfishly if there is gain to be had, even to the detriment of others.[1] A high-level goal in these systems is to design protocols that ensure the system will work well despite selfish behavior.

In this paper, we study the problem of selfish behavior in multi-hop wireless networks. The emergence of these networks is being driven by the rapid deployment of 802.11 networks and the advantages of relaying packets between nodes. In infrastructure rich areas, relaying can reduce dead spots, lower power consumption [31], and increase network capacity [19]. In rural or developing areas, multi-hop wireless networks can be deployed more readily and at lower expense than traditional wireless networks. Research examples of multi-hop networks include MIT's Roofnet [35], Microsoft's MUP [2], the Digital Gangetic Plains Project [8], and UCAN [27].

Selfish behavior is a concern in this setting because relaying packets for others consumes bandwidth and energy. Unlike traditional, wired LANs, nodes in these networks are often controlled by independent and potentially competing parties, e.g., nearby apartments [2, 35] or villages [8]. In the absence of any pressure to behave cooperatively, nodes have an incentive to *free-ride* by sending their own packets without relaying packets for others. This concentrates traffic through the cooperative nodes, which decreases both individual and system throughput, and may even partition an otherwise connected network.

Deployed routing protocols ignore the issue of free-riding. They simply assume that factors external to the routing protocol cause all nodes to cooperate. This incurs no overhead but unfortunately makes it trivial for a node to free-ride, e.g., by using a simple firewall rule to render itself indistinguishable from a node that lacks the wireless connectivity needed to relay traffic. Moreover, we show experimentally (Section 5.2) that free-riders can obtain substantial benefits. We should reasonably expect free-riding to become prevalent in all but the most benign situations.

Proposed solutions typically incorporate enough mechanism in the routing protocol to eliminate free-riding. This often involves some form of distributed accounting that allows each node to consume no more forwarding service than it provides. These solutions suffer from two serious drawbacks. They require infrastructure that seems unlikely to come about in practice, e.g., centralized clearance services [44, 34] or trusted hardware [11]. And they impose overly restrictive requirements on the system, e.g., uniform traffic rates among all node pairs [39].

Our goal is to combine the strengths of these two approaches while avoiding their weaknesses. Like deployed protocols, we assume that most (but not all) nodes will behave cooperatively. Like proposed solutions, we do not rely on trust alone but include mechanisms that actively discourage free-riding. The insight underlying this combination is that early users of a system are typically cooperative (as they try to get the system to work at all) while selfish behavior emerges when the user base grows [22]. Evolutionary game theory predicts that free-riding will not flourish if discouraged from an early stage [18].

Our solution is called Catch. It uses an existing majority of cooperative nodes to collectively discourage a minority of selfish nodes from free-riding. In game theory parlance, Catch assures that cooperation is an evolutionarily stable strategy. To achieve this, Catch uses novel techniques based on anonymous messages (in which the identity of the sender is hidden) to tackle two critical problems. First, Catch allows a cooperative node to determine whether its neighbors are free-riding, i.e., dropping packets that should be relayed. Second, it enables the cooperative neighbors of a free-rider to disconnect it from the rest of the network. These tasks can be accomplished even when cooperative nodes can communicate with each other only through potential free-riders. The result is that free-riding that previously succeeded is now deterred in a low-cost manner.

We have implemented and evaluated Catch on an in-building 802.11b testbed. This provides a realistic evaluation environment with the complex link quality factors that affect actual wireless systems. Real wireless conditions significantly complicate the implementation of robust mechanisms where nodes monitor the behavior of their neighbors. Yet they have received little attention in earlier work, which to our knowledge is exclusively based on simulation. We find that Catch is able to detect free-riding by individual nodes both quickly and with high accuracy. Its overhead is modest, roughly 24Kbps of control packets per node in our testbed, with no space overhead or cryptographic operations per data packet.

The rest of this paper is organized as follows. We describe our problem setting in Section 2, followed by our approach based on anonymous messages in Section 3. The Catch protocol itself is described in Section 4. Section 5 describes our evaluation based on the 802.11 testbed. We then report simulation results that analyze Catch across a broad range of parameters in Section 6. Finally, we present related work in Section 7 and our conclusions in Section 8.

2 Problem

We focus on selfish behavior, whereby a node gains at the expense of others, rather than malicious behavior, in which a node actively attacks others, e.g., by jamming its radio transmissions. Consider the simple example of a multi-hop wireless network in Figure 1. Here A may wish to send a message to C, either to communicate with C itself or because C serves as a gateway to additional nodes. Because A and C are not in each other's radio range, communication between then must rely on B. On the other hand, B may be interested in communicating via C but uninterested in obtaining any service from A. In that case, B may want to avoid the costs of forwarding packets for A.

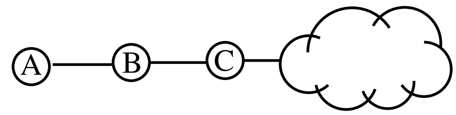

Figure 1: *An example multi-hop wireless network topology in which free-riding can take place.*

B can avoid these forwarding loads in two distinct ways: at the forwarding level and at the routing level. At the forwarding level, B can simply drop some or all of the data packets it receives for forwarding from A. At the routing level, B can refuse to send routing messages that acknowledge connectivity with A. Consequently, B will appear to be a "dead-end" from C's perspective and unreachable from A's, and so neither will ever request forwarding of it. This strategy, which we call *link concealment*, is broadly applicable and, to our knowledge, no existing wireless routing protocol or policing scheme counters it. Our protocol, Catch, prevents B from getting away with these selfish behaviors in the case that both A and C behave cooperatively. B would appear to be immune from adverse consequences for free-riding, because at best only A is aware of either of these behaviors (and it cannot communicate with C except through B), and only C can inflict any punishment on B. But we will see that this is not so.

Catch relies on three assumptions about nodes. First, most of them are cooperative in that they correctly run a protocol we define. A minority of nodes may be selfish and attempt to free-ride; we do not consider collusion amongst these nodes. Second, we assume omnidirectional radio transmitters and antennas, so that nodes can overhear nearby communications. This is true for common 802.11 hardware today. Third, nodes have an unforgeable identity. Such identities are not provided by current hardware but can be implemented by other means, e.g., using one-way hash chains [20] and imposing a startup cost for new identities.

Catch does not make any assumption regarding the routing protocol, traffic workload, or objectives of the nodes (such as bandwidth maximization or energy conservation). We believe that it works largely unchanged across these variables. We do not directly consider fairness issues but assume that a higher layer protocol decides what fraction of packets a node should relay for others. Catch can then be used to enforce that policy.

3 The Power of Anonymity

At a high-level, our approach is to use cooperative nodes to monitor for the presence of free-riders and to isolate them from the rest of the network. In this way, free-riding is no longer attractive. However, this approach requires us to tackle two problems, each of which is difficult or impossible to solve in the general case:

1. A node must be able to distinguish between selfish nodes that deliberately drop packets and cooperative nodes that simply do not receive them due to wireless transmission errors. It must do this from afar, even though packet reception events are not externally observable.

2. When a node detects a free-rider, it must be able to signal all of the free-rider's neighbors so that they can collectively isolate it. This must happen even when the only path to those neighbors is through the free-rider itself (which can simply refuse to forward messages that are not in its interest).

We show that anonymous messages, in which the receiver cannot determine the identity of the sender, can be combined with the broadcast nature of wireless to address both problems. This building block was first used in *Cocaine* [40]. Anonymous messages can be provided for most current 802.11 hardware by scrubbing the source MAC address on packets [7]. This forces would-be free-riders to engage in sophisticated games with signal strength measurements if they are to infer the sender. For now, we assume that anonymity can be provided and return to the impact of signal strength hints in Section 5.

3.1 Anonymous Challenges and Watchdogs

To distinguish deliberate packet dropping from wireless errors, we compare an estimate of the true connectivity of a node with its observed forwarding behavior. We use a watchdog [29] to observe the forwarding behavior of a *testee* node that is being tested for selfish behavior from a *tester* node that is assumed to operate correctly. (We use the terms *tester* and *testee* in these roles throughout this paper.) The watchdog relies on the broadcast nature of wireless transmissions. After a node sends a packet to a neighbor for relaying, it can listen to the wireless medium to observe whether the packet is forwarded by the neighbor. It can thereby build up an estimate of the neighbor's forwarding behavior over time.

It is more difficult to remotely estimate the true connectivity of a node. To do so, we develop an *anonymous challenge message* (ACM) sub-protocol as follows. Observe that even a selfish testee must depend on at least one of its testers to forward its packets if it is to stay connected. Call this tester the gateway. Let the gateway regularly but unpredictably send an anonymous challenge to the testee for it to rebroadcast; the gateway refuses to forward packets for the testee if it does not overhear the rebroadcasts (since it believes the testee is not connected or is free-riding). Now consider that all other testers with connectivity to the testee are also sending it anonymous challenges, requiring that they be rebroadcast. Because the testee cannot differentiate gateway challenges from other challenges, it must rebroadcast them all or risk losing connectivity to the gateway. This allows the other testers to estimate their connectivity to the testee. They then compare this to the observed forwarding behavior and infer deliberate packet dropping if there is a discrepancy. In practice, the estimates of connectivity and forwarding are statistical and only recent estimates are compared to allow for real wireless losses.

The ACM protocol is difficult to undermine even with weak anonymity because the likelihood of correctly handling a series of challenges decreases exponentially over time. Without breaking the protocol, a testee has only two options to avoid being flagged as deliberately dropping packets. First, it can be honest and reveal its true connectivity to its neighbors and forward their packets. This is what we desire. Second, it can selfishly drop both challenges and data packets in equal amounts and appear to be poorly connected to *all* its neighbors. But this is a counter-productive strategy. Because the challenges are anonymous they will be dropped independently of their source, and so data packets must also be dropped independently of their source to match. This forces the selfish node to drop and retransmit even its own packets, needlessly consuming its own resources. We note that the ACM protocol is compatible with nodes that sleep for power management, effectively dropping all packets. These nodes neither contribute to the network nor consume its resources, which we consider acceptable behavior. The ACM protocol also has the effect of discarding asymmetric links as does the 802.11 MAC.

3.2 Anonymous Neighbor Verification

Once a tester detects free-riding, it informs all other testers of the free-rider, so that they can simultaneously isolate it. This is necessary: if testers independently break connectivity with the free-rider, they only help the free-rider by reducing its forwarding burden while leaving it able to send its own packets through other testers. The challenge is to inform the other testers even though the only path to them might be via the free-rider, who may discard any incriminating information.

We define an *anonymous neighbor verification* (ANV) sub-protocol to allow a tester to reliably inform the other testers when the testee misbehaves. It operates in two phases. In the first ("ANV Open") phase, all testers become aware of each other via the testee: each tester sends a cryptographic hash of a randomly generated token to the testee for it to rebroadcast, and other testers take note when the rebroadcast happens. As before, anonymous messages are used to prevent the testee from selectively excluding testers. If the testee does not rebroadcast these messages, the testers assume that it lacks connectivity or is free-riding and do not relay packets for it.

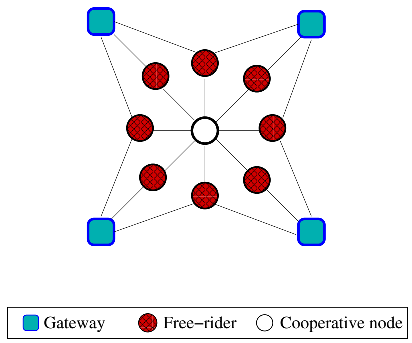

Figure 2: *An example topology to illustrate the use of* Catch*. The lines connect nodes that can directly communicate. (This is done to simplify the illustration; in reality, wireless connectivity is not binary but varies over a range [3].)*

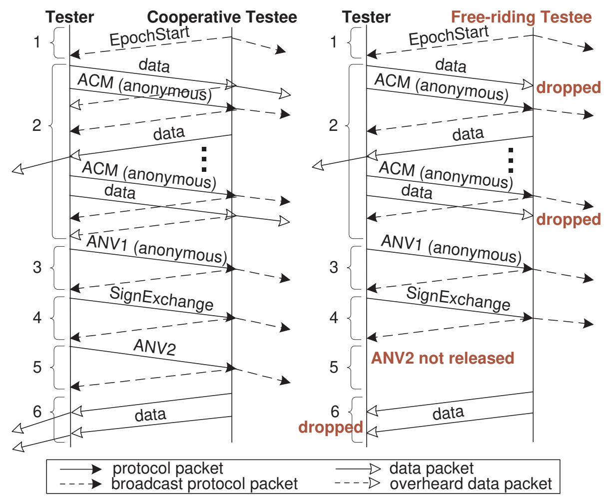

Figure 3: *Protocol flow. Packet exchange between a tester and a cooperative (left side) or free-riding (right side) testee. Numbers on the left of the time sequence correspond to the protocol steps.*

In the second ("ANV Close") phase, each tester releases its token to the testee only if the testee has behaved well, as determined by the ACM protocol. The testee rebroadcasts this token. If the hash of the received token matches one of the hashes collected during the first phase, other testers know that this particular tester is satisfied; the original token can only be released by the tester who encrypted it because it is computationally hard to invert the hash. If a tester does not eventually hear all of the tokens it expects based on the first phase, it concludes that another tester is signaling the presence of a free-rider by refusing to release its token. The free-rider is then isolated by all testers. Note that it is crucial that failure of the testee be signaled by the *absence* of a message to prevent the free-rider from blocking the signal, as it could with a more straightforward positive signaling mechanism.

We make two further observations. First, as before, dropping messages in the first phase to exclude particular testers and their data packets is unlikely to succeed. This is because the likelihood of correctly matching anonymous messages to testers decreases exponentially over time. Second, interference in the second phase of the sub-protocol by the testee is clearly unproductive because it can only lead to its isolation.

3.3 Example

We use an example to illustrate the power of the combined protocols. In Figure 2, a cooperative client is completely surrounded by free-riders. Without Catch the client cannot communicate with any of the gateways because the free-riders ignore its packets. With Catch, the client uses the ACM protocol to determine that it is in fact connected to the selfish nodes, and the watchdog to verify that its packets are not being dropped. If they are, the client uses the ANV protocol to inform the gateways, which isolates the free-riders. The threat of this punishment deters free-riding. Further, while we leave the issue of collusion for future work, Catch works for this topology even if the selfish nodes collude. This suggests a degree of collusion-resistance in the design.

4 The Catch Protocol

Catch builds on the anonymous techniques above, adapting them for use in real, wireless networks.

4.1 Overview

Catch operates as a sequence of protocol epochs run between a *testee* node and its neighbors, who act as *testers*. Figure 3 provides two illustrations of the per-epoch protocol steps, one when the testee is cooperating and the other when it is free-riding.

Each epoch consists of the following steps:

1. *Epoch-Start.* The testee broadcasts an EpochStart packet that includes its identity and an epoch identifier. Nodes that receive this request participate as testers for this epoch.
2. *Packet Forwarding and Accounting.* Testers run a watchdog [29] to count the number of their data packets that were correctly relayed. Note that the watchdog allows the testers to check for packet reordering (to force TCP backoff), corruption, or misdirection. Simultaneously, testers run the ACM protocol to estimate true connectivity. This involves sending anonymous challenges and counting their rebroadcasts; the data packets themselves are not anonymous.

3. *Anonymous Neighbor Verification Open (ANV1).* Each tester "opens" the two-phase ANV subprotocol (Section 3.2) by sending an anonymous packet containing a nonce (to prevent replay attacks) and a hashed token to the testee for rebroadcasting.
4. *Tester Information Exchange.* Each tester compares the fraction of its data packets that it overheard and the fraction of its anonymous challenges that it heard reflected. It obtains a one-bit ("sign") result depending on which is greater: 0 for challenges and 1 for data packets. It then sends its sign bit and identity to the testee for rebroadcasting.
5. *Epoch Evaluation and ANV Close (ANV2).* Each tester determines whether the testee is operating correctly using its observations and the sign bits from other testers. This is done with a pair of statistical tests described in the next subsection. If both tests pass (and the testee correctly rebroadcast the tester's sign bit), the tester releases its token. Otherwise, it withholds its token.
6. *Isolation Decision.* An epoch fails for a tester if it withholds its token or it does not receive all expected tokens. If too many epochs fail too quickly (Section 4.3) then the tester decides that the testee is free-riding and punishes it by dropping its packets for a fixed number of epochs. By virtue of the protocol, all testers decide to punish a free-rider at (nearly) the same time, so that it is isolated.

We increase the likelihood of all testers seeing all control packets in two ways. First, we use retransmissions if a tester does not hear the rebroadcast. Second, we use cumulative broadcasts, where the testee sends all of the information it has received on every transmission.

4.2 The Per-Epoch Tests

Each tester applies two statistical tests per epoch to determine whether a testee is behaving correctly. Each test is designed to be sensitive to distinct selfish strategies. The key challenge in both is to avoid mistaking volatile wireless conditions for misbehavior.

One selfish strategy is to drop packets from a particular tester in the hope that the consensus across neighbors will be that the free-rider has passed the epoch, since all other testers should find its behavior acceptable. To detect this, each tester compares observed forwarding and true connectivity estimates for the last three epochs using the z test [30]. We found that high confidence levels (99% and above) coupled with using measurements from multiple epochs provides a good balance between quick detection of free-riding and a low rate of false positives.

The second selfish strategy is to uniformly drop some fraction of the packets received from each tester, making it hard for any one of them to conclude that free-riding has taken place. To detect this, we employ the sign test [30] using the sign bits exchanged by all testers. This test is based on the idea that the perceived forwarding and connectivity rates should have identical means if the testee is not deliberately dropping packets. Thus, random fluctuations in each epoch should yield about as many results in which one exceeds the other as the opposite. Each tester accumulates the one-bit results for all epochs in which it has participated, and applies the sign test to decide if the balance is reasonable.

4.3 The Isolation Decision

Isolation of a testee is decided by all testers in parallel. Each maintains a small history of per-epoch test results, represented as a three state finite state automaton (FSA) that moves to the right when an epoch fails and the left when an epoch passes. If the FSA falls off the right edge, the testee is isolated.

While it might seem that this scheme allows a node to free-ride for at least half of the epochs, the fact that the per-epoch test results depend on packet accounting data aggregated over the previous three epochs prevents this: free-riding in any one epoch impacts the tests for three consecutive epochs, and is likely to lead to multiple failed tests. We more fully explore this issue in Section 6.3.

4.4 Protocol Fail-safes

Because Catch is designed to operate when some nodes act in a selfish manner, we are as concerned about what happens when the protocol is not followed as when it is. In Appendix A we provide a short analysis by message type that shows that selfish nodes cannot undermine the protocol in the absence of collusion.

5 Experimental Evaluation

This section describes our experiments with Catch on an 802.11b testbed. This allows us to test how well Catch works in wireless environments that exhibit complex packet loss behaviors [24].

5.1 The Testbed

Our testbed is composed of 15 PCs equipped with 802.11b that run Linux 2.4.26. We use NetGear MA311 PCI network adapters (Prism 2.5 chipset), operating in the ad-hoc mode on channel 1 using the *hostap* driver. Each node also has a wired Ethernet interface to facilitate remote management of the experiments.

The testbed is located on a single floor of an office building, as shown in Figure 4. The building has its own

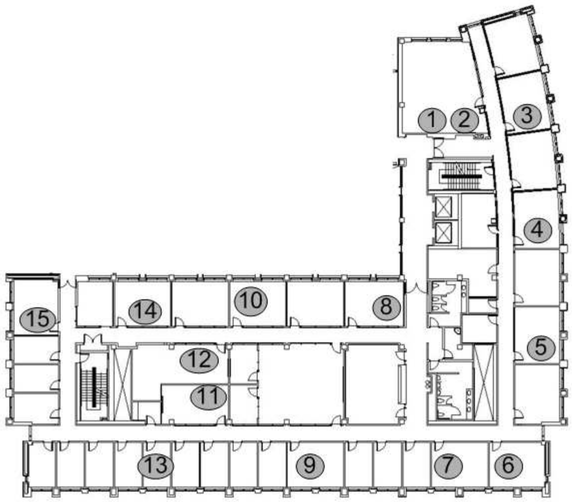

Figure 4: *Our wireless testbed, consisting of fifteen 802.11b nodes. The node locations are marked with circles. Horizontally, the building is 184 ft. long.*

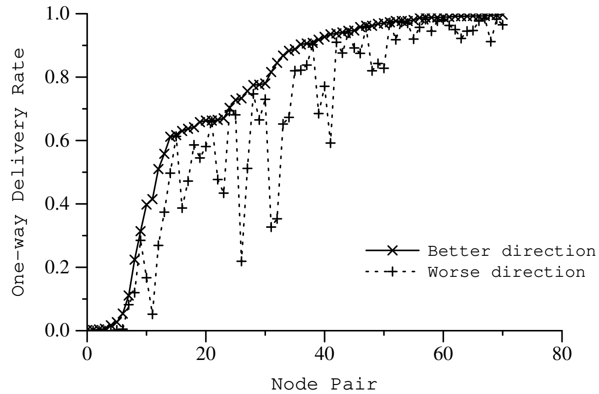

Figure 5: *For each node pair in the testbed, the fraction of sent packets successfully received in each direction. There are 105 pairs total in the testbed. Only node pairs with a non-zero delivery rate between them in at least one direction are shown.*

dense deployment of wireless access points, including ten on the same floor as our testbed, some of which compete with us on channel 1. Such a setting is noisy, but realistic [3].

Our system exhibits well-known characteristics of wireless networks, including error rates that are not a simple function of distance, that are strongly asymmetric, and that vary widely over time. Figure 5 gives a static summary of these effects. It shows the average one-way delivery rate in each direction for each pair of nodes that were able to communicate at all. To compute these rates, each node broadcast 500 1000-byte packets over two minutes. The other nodes counted how many of those packets they received. The figure shows a wide range of delivery rates rather than a binary state of connectedness, which is consistent with prior results [3, 43]. The diameter of our network is between 3 and 5 hops, depending on the threshold of link quality at which two nodes are considered connected.

5.1.1 Catch Implementation

We implemented Catch at user-level using the Linux *netfilter* framework to monitor and manipulate the packets sent, received, and forwarded by a node. The watchdog component of Catch also needs to overhear all packets sent by the node's neighbors regardless of their intended destination. To capture these packets, we operate our wireless network adapters in promiscuous mode and use the Linux pcap framework. The Catch protocol itself is written in ruby and is completely independent of the underlying routing protocol.

One complication is that the watchdog mechanism needs to account for 802.11 MAC-level retransmissions. To see this, consider a tester judging whether the testee forwarded a particular data packet. The quality of the link between the testee and the recipient determines the number of retransmissions done by a cooperative testee. This in turn changes the probability that the tester will overhear the transmission. To correct for this recipient-based variation, we measure the data forwarding rate using only the first transmission as indicated by a bit in the 802.11 MAC header. A complete implementation of Catch would also check that retransmissions are handled consistently to close a secondary loophole. We have not done so yet.

We use the following parameters values for our experiments; simulations suggest that Catch is not highly sensitive to the exact choices. The length of an epoch is set to one minute. The confidence interval for the z test is 99.999%, and that for the sign test is 99.995%. (Both experiments and simple analysis showed that very high confidence values are most effective.) There are fifteen anonymous ACM messages per epoch, each of which is 1500 bytes, the MTU (maximum transmission unit) size of our network adapters. The loss rate for smaller data packets (such as TCP acknowledgements) can be less than that of the ACM messages. To verify forwarding behavior, our implementation checks that the loss rate for data packets is less than that for ACM messages.

5.1.2 Multi-Hop Performance

We first show the potential benefit of relaying packets by comparing the performance of a single, centrally located access point (AP) setup to that of multi-hop routes. To do this we transfer a large file from one node, which acts as the AP (node 8 in Figure 4), to four client nodes (nodes 4, 6, 9 and 14). Each client downloads a 600KB file ten

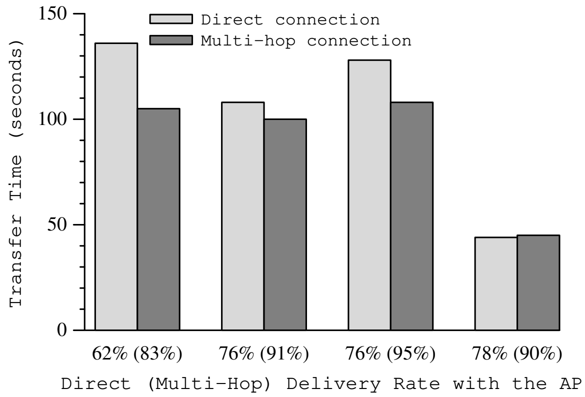

Figure 6: *Time to transfer 6MB from node 8 in Figure 4 by four other nodes via direct and multi-hop connections. The x-axis label gives the delivery rate of the direct connection with node 8, with the delivery rate for the multi-hop path in parentheses. The client nodes, from left to right, are 4, 6, 14, and 9.*

times. In one set of experiments, the clients communicate directly with the AP. In the other, they use multi-hop routes via a single intermediary node, over paths 4:5:8, 6:7:8, 14:10:8, and 9:10:8. We use static routes between nodes to factor out effects that stem mainly from the routing protocols; wireless routing protocols are an open area of research [14].

Figure 6 shows the results. The x-axis labels give the delivery rate of the direct links, averaged over both directions. The parenthesized numbers give an estimate of the quality of the two-hop path, computed as the product of the delivery rate of the individual links. In total, the use of multi-hop paths reduced download time by 16%, with per-node benefits ranging from 30% to -2%. The better performance of the multi-hop routes is due in part to the lower packet loss rates they enjoy. De Couto *et al.* have studied these issues in more detail [15, 14].

5.2 The Impact of Free-riders

We now consider the performance impact of free-riding, both as benefits to the free-riders and as costs to the cooperative nodes. We do this by contrasting the per-node throughput achieved in a fully cooperative network with those achieved when some nodes are allowed to free-ride.

In this experiment, we randomly selected 3 nodes as free-riders. All nodes were trying to download randomly selected files from randomly selected servers. Figure 7 illustrates the average amount of data transferred under the two scenarios: “Free-riding Discouraged,” which results in all nodes behaving cooperatively, and “Free-riding Ignored,” where free-riders simply do not relay packets for cooperative nodes. Both scenarios were run for 35 minutes. The two bars in each scenario average the per-node results for twelve nodes that acted cooperatively and for the three free-riders. The data illustrates two key points.

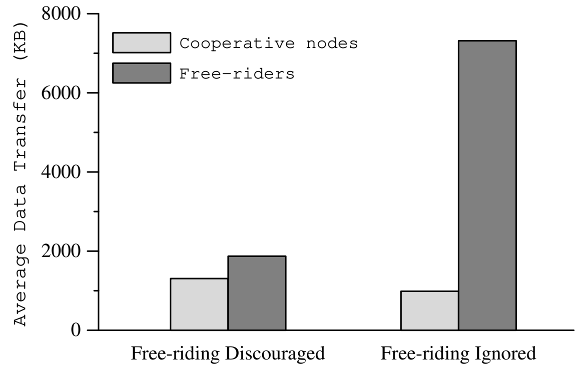

Figure 7: *Average amount of data transferred per node when free-riding is discouraged and when it is not. None of the nodes free-ride in the former. Nodes 7, 14 and 15 free-ride in the latter.*

First, there is a very large incentive to free-ride: the free-riders improve their throughput by 400% relative to when they are forced to cooperate. This indicates that there is considerable potential motivation for nodes to behave selfishly in these environments if they can do so without retribution. Second, the improved situation for the free-riders comes at the expense of cooperative nodes. The performance of the cooperative nodes is decreased by 25% when 20% of their fellow nodes selfishly misbehave. While this is only a single example, it clearly demonstrates the need to incorporate protection against free-riding in routing protocols.

5.3 Catch Evaluation

In this section we evaluate the effectiveness of Catch.

5.3.1 Detecting Free-riders

Our first experiment measures the speed with which Catch detects free-riding. To construct a base case, we selected triplets of nodes such that both the first and the third node had a reasonable (>75%) delivery rate to the second node. The second node was configured to act as a free-rider that randomly dropped a fraction of the packets it received for forwarding. We experimented with different drop rates; Drop rates less than 100% mimic a situation in which the free-rider tries to evade detection by appearing to be a cooperative but poorly connected node. The first node downloaded randomly selected files ranging from 1KB to 3MB in size from the third node. The request and response traffic was relayed through the second node. Five download sessions ran in parallel so that even in the presence of a high drop rate and TCP back-off dynamics, a minimum amount of traffic (roughly ten packets per epoch) is generated for the statistical tests.

Figure 8 presents the results. The line “Drop packets from both” corresponds to the case when the free-rider drops packets from both neighbors. It shows the average number of epochs required to detect a free-rider for

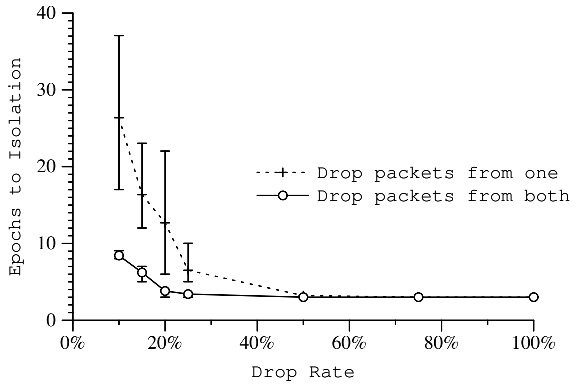

Figure 8: *The number of epochs required to detect free-riders in the testbed versus the fraction of packets a free-rider dropped. Each point is the average of 10 experiments. Vertical bars represent the inter-quartile range.*

varying drop rates. Catch reacts quickly to free-riding, and its reaction time decreases with drop rate. Detection is almost immediate for very high drop rates; recall from Section 4.3 that at least three epochs must fail before isolation. Even at the low drop rate of 10%, Catch isolates the free-rider in under 9 epochs.

The curve "Drop packets from one" shows the results for the case where the free-rider dropped packets only for the client. This evaluates whether a single victim can cause the free-rider to be isolated. We find that for high drop rates the detection speed is just as fast as the previous case. It is slower at lower drop rates, but even at the low drop rate of 10% the average detection time is less than 30 epochs. Thus, a free-rider that persistently drops packets of just one neighbor at a very low rate is eventually caught and punished.

5.3.2 False Accusations

We next check that the rapid detection of free-riders does not come at the cost of falsely accusing cooperative nodes of free-riding. We ran two five hour experiments in which all nodes were cooperative. Each node repeatedly downloaded files (as before) from randomly chosen servers. This workload is high enough to saturate our network, stressing the accuracy of inference and increasing the probability of false accusations. We observed no false positives in the first experiment and a single false positive in the second. It is difficult to estimate the true rate of false accusations from this because they are so rare, but nevertheless we find it encouraging.

5.3.3 Coordinated Isolation

We now evaluate whether wireless conditions hinder the ability of the testers to simultaneously isolate a free-rider.

We randomly selected three (20%) nodes as free-riders that dropped all the packets they received for forwarding. All nodes executed a workload similar to the one in

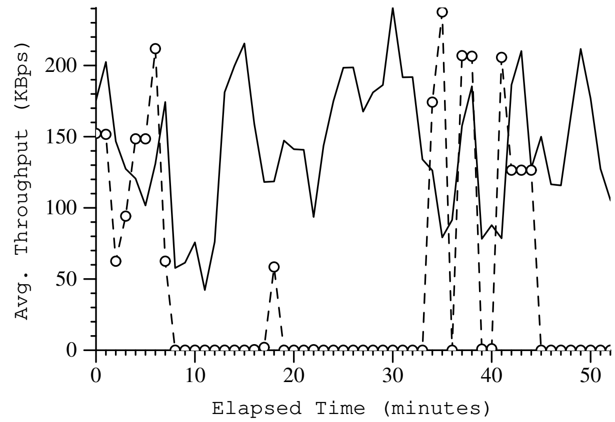

Figure 9: *Average throughput of cooperative nodes (solid line) and free-riders (dashed line) as a function of time. Throughput was calculated using one minute intervals. There were three free-riders. The punishment interval is 30 minutes.*

the previous section with the exception that nodes only selected the cooperative nodes as file servers. We then measured the throughput obtained by the free-riders. It should be zero if coordinated isolation was successful.

Figure 9 plots the average throughput obtained by the free-riding and cooperative nodes. It shows that the cooperative nodes successfully shut out the free-riders. Roughly eight minutes into the experiment, all the free-riders were identified and isolated. Though not shown in the graph, the spread of time over which different neighbors of a free-rider started isolating it was two minutes. The free-riders were allowed to send traffic again after the punishment interval of 30 minutes. The average throughput of the free-riders appears to recover before 30 minutes because different free-riders were isolated and released at different times.

5.3.4 Protocol Overhead

We report on the overhead of Catch in this section. We have made no attempt to optimize the protocol because its requirements are already modest.

Consider the activity for a pair of neighboring nodes in an epoch, both playing the role of tester and testee. The packet overhead of Catch comes from its messages, which have different sizes and frequencies: StartEpoch (40 bytes), ACM challenges and responses (1500 bytes, 15 times per epoch), ANV open and close (100 bytes), and sign exchanges (40 bytes). These packets come to a total of 0.6 packets or 758 bytes per neighbor per second. Our testbed has fewer than four well-connected neighbors per node on average, which means that the protocol overhead is less than 2.4 packets per second or 3 KBps per node. This is 3% of the 100 KBps that the honest nodes got on average in Figure 9. The overhead would be even lower for the newer and faster 802.11a/g.

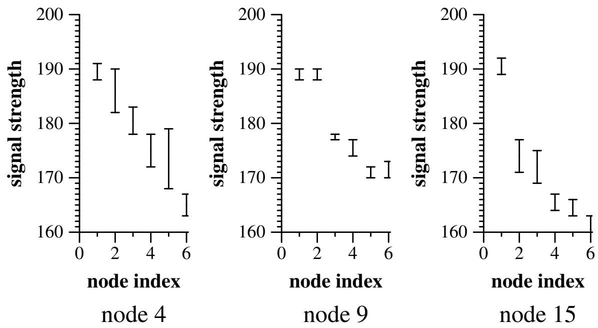

Figure 10: *The spread of received signal strength at Nodes 4, 9 and 15 in our testbed. The y-axis represents the magnitude of the signal reported by the hardware. The bars represent the range in which 90% of the packets from a neighboring node fall.*

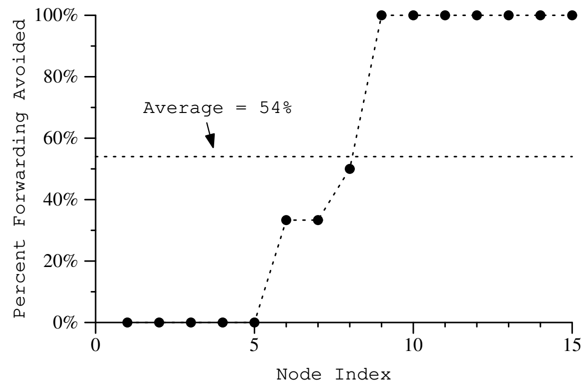

Figure 11: *The fraction of forwarding load avoided if a node adopts a signal strength cheating strategy. (We assume forwarding load is proportional to the number of neighbors.)*

We found the processor consumption of Catch to also be very reasonable. Informally observed using *top* during our experiments, it took at most 10% of the CPU on Pentium-IV 3 GHz nodes. Much of this is an artifact of our user-level implementation. Each packet that passes through the local machine or is promiscuously overheard crosses the user-kernel boundary at least once. In fact, before moving to a PC-based testbed for OS reliability reasons, we had successfully experimented with Catch on a testbed composed of 10 iPAQs.

5.3.5 Compromising Anonymity

In this section we study the potential leverage of signal strength attacks on anonymity. We show that even in its present form Catch is useful in protecting the cooperative nodes and is by far preferable to doing nothing. Taking specific steps in Catch to discourage signal strength based cheats is the subject of future work.

At the MAC level, anonymity is a reasonable assumption, since it is possible to send packets with an arbitrary source address and contents using commonly available 802.11 hardware [7]. At the physical level, however, strong anonymity cannot be guaranteed against a determined adversary: the source of a packet might be estimated, or at least classified, from the wireless signal strength or direction.

Signal strength cheats are a level of escalation beyond the selfish misbehavior we have defended against thus far. Free-riding using signal strength measurements is not a simple matter of installing a firewall rule, but requires changes to the network interface driver. Our hardware cannot give information about signal source direction, nor can any commodity hardware (fitted with an omnidirectional antenna) of which we are aware.

Catch provides protection against such cheats because the received signal strength from an individual neighbor varies over a range of values. When the ranges of multiple neighbors overlap it becomes impossible to accurately distinguish among them. Empirical reports of wireless network conditions [42, 43, 24] and localization schemes based on received signal strength [5] illustrate the difficulties of using signal strengths. As examples, Figure 10 shows the spread of received signal strength at three nodes in our testbed.

To better understand the overall threat, we experimented with a cheater that uses signal strength to differentiate among its neighbors. The cheater listens to data packets for a short period of time, measuring their signal strengths and sources. It then chooses a signal strength threshold at which to drop incoming packets. It relays packets and appears cooperative to neighbors whose packets arrive with strengths above the threshold. It drops packets below the threshold to appear to be a legitimate non-neighbor to all other nodes.[2] Using this procedure, a cheater may end up cooperating with between just one and all of its legitimate neighbors. Of the nodes in Figure 10, Node 4 is forced to cooperate with all of its neighbors, Node 9 with only two of them, and Node 15 with only one of them. (Peripheral nodes that can uniquely identify a neighbor do not present a major threat as such nodes are not expected to relay packets.)

Figure 11 shows the benefits of this attack in our testbed. For each of the 15 nodes we plot the fraction of forwarding traffic that would be avoided, assuming that forwarding loads are proportional to the number of neighbors, or zero if a cheater manages to establish only a single neighbor. We conservatively assume that when a cheater identifies a subset of its neighbors, one of the nodes in the subset is capable of forwarding packets for it; otherwise, the cheater needs to admit connectivity to other neighbor(s). Just under half the time a cheater can escape forwarding entirely, while just over half it avoids none or only a modest amount. Of course, if no protocol is run to protect against cheating, all nodes can cheat 100%, leading to a tragedy of the commons.

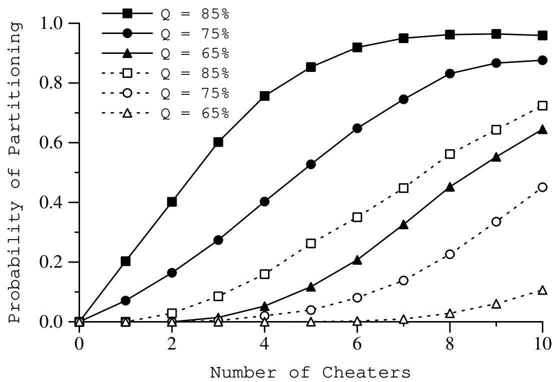

Figure 12: *Probability that the cooperative nodes are partitioned versus varying numbers of (randomly chosen) cheating nodes when running with (dotted lines) and without (solid lines)* Catch. *Under* Catch *the cheaters use a signal strength based cheating strategy. Only links with delivery rates at least Q are considered useful.*

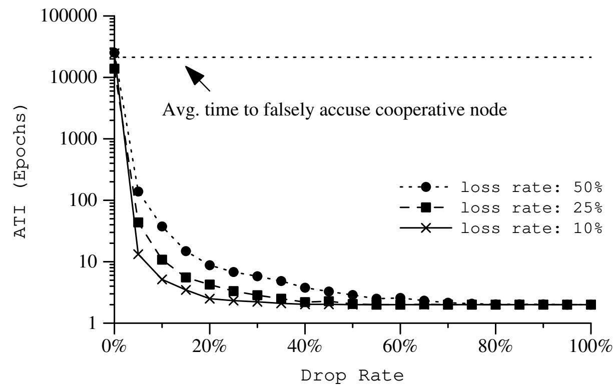

Figure 13: *Average time to isolation versus drop rate, for various background network loss rates.* (Y-axis on a log scale.)

Even though a cheater may expect to reduce its forwarding load by about half using signal strength information, Catch still helps the cooperative nodes. Figure 12 shows that Catch greatly improves connectivity for those nodes, relative to taking no measures against cheating. It plots the probability that a (randomly selected) set of such cheaters would partition the cooperative nodes when running with and without Catch. Because Catch forces many cheaters to admit to multiple neighbors, and so to be available for packet forwarding, it significantly reduces the odds that the network is partitioned. For example, when 20% (3) of the nodes cheat, that probability is lowered from about 60% to about 10% when using the highest quality links. At a 75% link delivery rate threshold, the odds of a network partition are reduced from about 30% to zero. Of course, these results are specific to our testbed; in general, the extent of protection provided by Catch depends on the degree of overlap between the signal strengths of different neighbors. We are currently extending Catch to mitigate such attacks by having testers vary their signal strength as part of the testing.

6 Simulation and Analysis

We now extend our analysis of Catch using simulation.

6.1 Simulation Testbed and Metrics

We built a simulator to generate packet loss and reception counts for each epoch and to drive the protocol state machine. The simulator does not model the details of packet delivery. The protocol state machine is parameterized by the neighborhood topology, its loss rates, and the z and sign statistical test parameters. We focus on packets that are subject to Catch's statistical tests and ignore other (control) packets. Our base setting includes a single free-rider with six neighbors. The epoch duration in the simulations is one minute. We set the confidence levels for the z and sign tests to 99.999% and 99.995% respectively. Results from the simulator showed that these values achieved the best overall tradeoff between detection speed and false positive rate.

To assess the effectiveness of Catch, we use *Average Time to Isolation* (ATI) as the metric. ATI is measured in units of epochs. An ideal policy would exhibit ATI values of one for nodes that free-ride (at any rate), and infinite ATI values for those that do not.

6.2 Physical Environment Effects

We first evaluate Catch's robustness to two characteristics of the physical environment: packet loss and network density. To model free-riding, we use a straightforward strategy in which the free-rider drops packets randomly with fixed probability. Because the packet losses due to the wireless network are also modeled as a random process, this drop strategy is arguably difficult for our statistical tests to detect.

6.2.1 Packet Loss

We would expect higher wireless loss rates to make it more difficult to detect free-riding. Figure 13 shows ATI results as a function of drop rate for three different background network loss rates. Each data point shows the average of 40 runs. When there is no free-riding (the y-axis), there is a large isolation time – an average of around 26,000 epochs (about 18 days). These times fall steeply as the drop rate grows, to under 10 epochs for drop rates of 10-20%. The results for loss rates in the range of 10%-25% are in line with those observed in our testbed (Figure 8), except that the homogeneous link qualities in the simulation environment result in much longer false accusation times. Thus, the impact of high

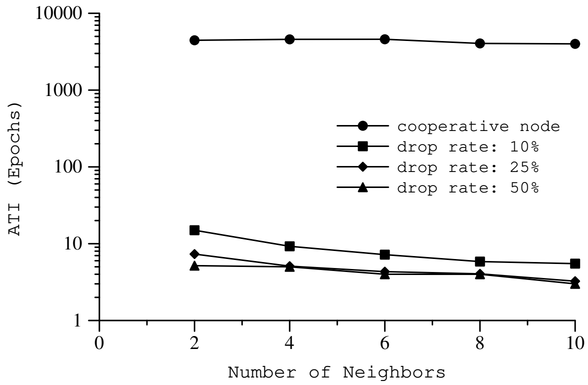

Figure 14: *Average time to isolation versus number of neighbors.* (Y-axis on a log scale.)

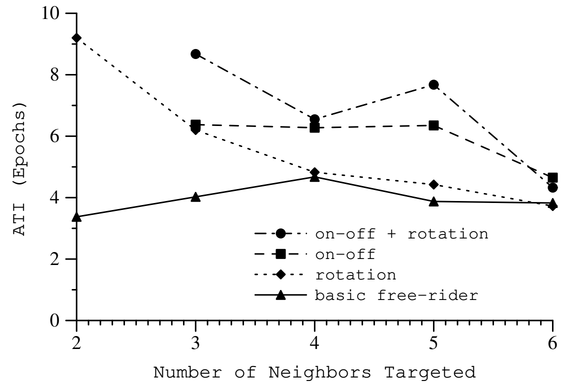

Figure 15: *The time to isolation for customized free-riding strategies. The free-rider directs all misbehavior in a single epoch to the number of neighbors given on the x-axis.* (Total cheat rate = 20%. Network loss rate = 20%.)

wireless loss rates on Catch is quite small. Even at a network loss rate of 50% Catch isolates a free-rider who drops 25% of the packets it needs to forward in seven epochs on average, which is only four epochs more than the fewest possible.

6.2.2 Network Density

We would expect Catch to perform better in denser networks because larger neighborhoods are more likely to make correct statistical decisions. Figure 14 examines the impact of the number of neighbors on detection and false accusation times. We show results for a cooperative node (the top line) as well as for free-riders at drop rates from 10-50%. Increasing the number of neighbors from six to ten yields a small decrease in the time to detect free-riders, as might be expected: already at 6 neighbors there is little room for improvement. More surprisingly, reducing the number of neighbors by a factor of three, to only two, increases detection time by only a few epochs. Additionally, the rate at which cooperative nodes are falsely accused is essentially unaffected over the entire range. Thus, Catch seems to be robust, working well in both high and low density networks.

6.3 More Sophisticated Cheaters

Thus far, we have analyzed a simple drop model in which the free-rider randomly drops packets it is meant to forward. We now use our knowledge of the statistical tests to construct packet dropping variations that target potential weaknesses. While we cannot prove the negative result that there are no strategies that might be effective against Catch, we can show that these customized strategies yield only very limited success.

One variation is targeted free-riding, in which the free-rider drops packets from a time-varying subset of neighbors, rather than uniformly from all. This stresses the z test in Catch, whereas we know that the basic free-rider is most often detected by the sign test. We call this approach "rotation." A second variation attacks the isolation decision process. Since three consecutive failed epoch tests are required to isolate a node, a free-rider may attempt to escape isolation by dropping packets on, say, alternate epochs. We call this the "on-off" strategy. Finally, both attacks may be used at once.

Figure 15 plots the number of epochs to isolation for these strategies against the number of nodes targeted, for the difficult environment where the loss rate is as large as the drop rate. (Both were set to 20%.) The graph suggests that these custom-built strategies are only very modestly successful. The most effective strategy for the free-rider is to obtain its overall average drop rate of 20% by dropping 60% of the packets from two of its six neighbors, while rotating that pair each epoch. Using that strategy, the free-rider is isolated in nine epochs on average, compared to five epochs for the base free-riding strategy.

As another variation of the basic free-riding model, we experimented with free-riders that drop packets in a deterministic pattern, rather than randomly. The threat here is that the reduction in variance will help free-riders avoid detection. In fact, the opposite happened: Catch was more effective.

6.4 Assessing Effectiveness

To complete this section, we consider how much better it might be possible to do than Catch. This is a difficult question to answer. We address it by comparing Catch to an unrealistically powerful alternative, the *Detection Oracle*, that serves as an informal upper bound on what might be possible by any technique.

The Detection Oracle hears all packet transmissions everywhere in the network, without loss, and so has reliable knowledge of all externally visible events. Addi-

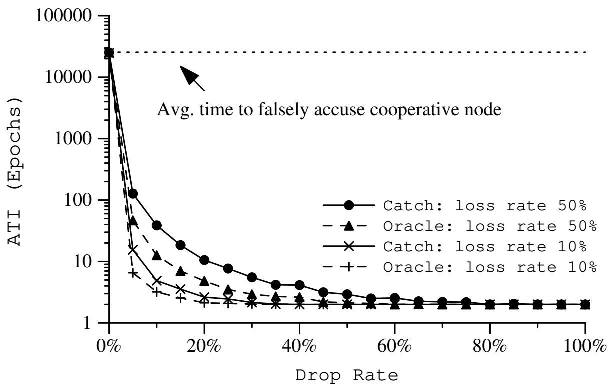

Figure 16: *Comparison of the time to isolation with* Catch *and the Detection Oracle as a function of drop rate for 10% and 50% network loss rates.* (Y-axis on a log scale.)

tionally, it retains infinite history information, enabling it to apply the Catch statistical tests over this maximal pool of data. In contrast, the nodes in any real system have only imprecise information (due to losses), each one is directly aware of only a subset of the global information, and history information must be devalued due to the changing environment.

Figure 16 compares the Detection Oracle with Catch. It suggests that Catch does nearly as well as possible. The oracle's advantage exceeds a five epoch reduction in detection time only in the case of high network loss rate (50%) and relatively low (5-25%) drop rates.

7 Related Work

Anonymous broadcast was first used as a protocol building block in the *Cocaine* protocol for auction between mistrustful parties [40]. In a manner similar to Catch, Cocaine combines this building block with one-way hash functions. We apply this approach in a different and practical setting, and our work also hints at the generality of the building block and the approach.

Catch belongs to the class of enforcement-based mechanisms that discourage free-riding through the fear of punishment. The watchdog part of our detection mechanism was originally proposed by Marti *et al.* [29]. It is our use of it in real networks and in conjunction with anonymity to detect misbehavior that is novel. Existing enforcement-based protocols [29, 10, 9] rely on reputation spreading to deal with cheating nodes. This requires global flooding, while Catch limits information spread to single-hop neighborhoods. Moreover, simple flooding requires network redundancy as selfish nodes will not forward incriminating reputation packets. Catch uses anonymity and one-way hash functions to reliably communicate with the neighbors of free-riders. Our use of one-way hash functions is similar to Hu *et al.*'s work on secure routing in wireless networks [20, 21].

Incentive-based approaches discourage free-riding by making cooperation more attractive. Nodes accumulate virtual currency by forwarding for others, which they can then use for sending their own packets. Examples include Nuglets [11], Sprite [44] and priority forwarding [34]. These schemes rely on a trusted central authority or tamper-proof hardware to ensure the integrity of the currency, and to redistribute wealth so that even nodes that are not in a position to forward for others can send their packets. In contrast, the operation of Catch is completely distributed. Incentives also fail to encourage nodes with very little data of their own to send. This can lead to a disconnected network when light-senders are located at strategic points in the topology.

Finally, game-theoretic approaches formulate the forwarding decision such that forwarding at a certain rate becomes the Nash equilibrium [18] for the network. This means that deviation from the recommended forwarding behavior can only result in situations that are worse for the deviant node. Generous Tit-for-Tat (GTFT) is an example of such an approach [39]. Like GTFT, Catch relies on the mechanics of Tit-for-Tat by assuming cooperation and punishing free-riders. However, while GTFT requires knowledge about the utilities of all the nodes in the network, Catch relies only on information collected in the one-hop neighborhood of individual nodes.

8 Conclusions

We have presented Catch, a protocol to sustain cooperation in multi-hop wireless networks comprised of autonomous nodes. Catch is much more widely applicable than other proposed solutions, needing no central authority and placing no restrictions on workloads, routing protocols or node objectives. It uses novel strategies based on anonymous messages and statistical tests to detect free-riders with high likelihood and punish them with periods of isolation. *Anonymous challenge messages* are used to estimate true loss rates, even when dealing with untrusted and uncooperative nodes. *Anonymous neighbor verification* is used to compel a node to forward packets, even when the data being carried is contrary to its interests. While our application of anonymity and neighborhood watch are specific to the wireless domain, we expect that these techniques are general enough to be applicable in other domains.

We implemented Catch in Linux and performed what to our knowledge is the first evaluation of cooperative routing protocols in an 802.11 wireless testbed. We showed that Catch works well despite volatile wireless conditions and requires little bandwidth overhead (and negligible CPU overhead). In our experiments, free-riders are quickly isolated from the network (and more rapidly for more egregious drop strategies) and cooperative nodes are rarely accused of misbehaving. Simulations confirm this finding over a wide range of conditions. We quantified the impact of free-riding by showing that the presence of even a few free-riders can partition the network. In one experiment, their presence led to a 25% overall performance degradation for the cooperative nodes. We also explored the leverage of signal strength cheats, and found that even without any measure to actively thwart such cheats, Catch provides worthwhile protection. Extending Catch to defeat these strategies is part of our future work.

9 Acknowledgements

We thank Krishna Gummadi, Ed Lazowska, Hank Levy, Mike Swift, the anonymous reviewers, and Amin Vahdat, our shepherd, for their comments on earlier versions of this paper. Brian Youngstrom provided valuable assistance with setting up the testbed.

This work was supported in part by NSF (Grant No. ANI-0133495), Atheros Communications, and Microsoft Research.

References

[1] E. Adar and B. A. Huberman. Free riding on gnutella. *First Monday*, 5(10), Oct. 2000.

[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-2003-41, Microsoft, June 2003.

[3] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. A measurement study of a rooftop 802.11b mesh network. In *ACM SIGCOMM*, Aug. 2004.

[4] A. Akella, R. Karp, C. Papadimitrou, S. Seshan, and S. Shenker. Selfish behavior and stability of the Internet: A game-theoretic analysis of TCP. In *ACM SIGCOMM*, Aug. 2002.

[5] P. Bahl and V. Padmanabhan. RADAR: An in-building RF-based user location and tracking system. In *IEEE INFOCOM*, Mar. 2000.

[6] N. Baughman and B. N. Levine. Cheat-proof playout for centralized and distributed online games. In *IEEE INFOCOM*, Apr. 2001.

[7] J. Bellardo and S. Savage. 802.11 denial-of-service attacks: Real vulnerabilities and practical solutions. In *USENIX Security Symposium*, Aug. 2003.

[8] P. Bhagwat, B. Raman, and D. Sanghi. Turning 802.11 inside-out. In *HotNets-II*, Nov. 2003.

[9] K. A. Bradley, S. Cheung, N. Puketza, B. Mukherjee, and R. A. Olsson. Detecting disruptive routers: A distributed network monitoring approach. In *IEEE Symposium on Security and Privacy*, May 1998.

[10] S. Buchegger and J.-Y. L. Boudec. Performance analysis of the CONFIDANT protocol: Cooperation of nodes — fairness in dynamic ad-hoc networks. In *MobiHOC*, June 2002.

[11] L. Buttyan and J. Hubaux. Stimulating cooperation in self-organizing mobile ad hoc networks. *ACM/Kluwer Mobile Networks and Applications*, 8(5), Oct. 2003.

[12] B.-G. Chun, R. Fonseca, I. Stoica, , and J. Kubiatowicz. Characterizing selfishly constructed overlay routing networks. In *23rd IEEE International Conference on Computer Communications*, Mar. 2004.

[13] B. Cohen. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[14] D. De Couto, D. Aguayo, J. Bicket, and R. Morris. A high-throughput path metric for multi-hop wireless routing. In *MobiCom*, Sept. 2003.

[15] D. De Couto, D. Aguayo, B. A. Chambers, and R. Morris. Performance of multihop wireless networks: Shortest path is not enough. In *HotNets-I*, Oct. 2002.

[16] S. Floyd. Congestion control principles. IETF RFC 2914, Sept. 2000.

[17] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4), Aug. 1999.

[18] D. Fudenberg and J. Tirole. *Game Theory*. MIT PRess, Aug. 1991.

[19] P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, IT-46, Mar. 2000.

[20] Y.-C. Hu, D. Johnson, and A. Perrig. Secure efficient distance vector routing for mobile wireless ad hoc networks. In *IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, June 2002.

[21] Y.-C. Hu, A. Perrig, and D. Johnson. Ariadne: A secure on-demand routing protocol for ad hoc networks. In *MobiCom*, Sept. 2002.

[22] E. Huang, J. Crowcroft, and I. Wassell. Rethinking incentives for mobile ad hoc networks. In *ACM SIGCOMM PINS*, Sept. 2004.

[23] F. Kelly, A. Maulloo, and D. Tan. Rate control in communication networks: shadow prices, proportional fairness and stability. *Journal of the Operational Research Society*, 49:237–252, 1998.

[24] D. Kotz, C. Newport, and C. Elliott. The mistaken axioms of wireless-network research. Technical Report TR2003-467, Dartmouth, July 2003.

[25] K. Lai, M. Feldman, I. Stoica, and J. Chuang. Incentives for cooperation in peer-to-peer networks. In *Workshop on Economics of Peer-to-Peer Systems*, June 2003.

[26] S. H. Low, F. Paganini, and J. C. Doyle. Internet congestion control. *IEEE Control Systems Magazine*, Feb. 2002.

[27] H. Luo, R. Ramjee, P. Sinha, L. Li, and S. Lu. UCAN: A unified cellular and ad-hoc network architecture. In *MobiCom*, Sept. 2003.

[28] R. Mahajan, S. Floyd, and D. Wetherall. Controlling high-bandwidth flows at the congested router. In *International Conference on Network Protocols (ICNP)*, Nov. 2002.

[29] S. Marti, T. J. Giuli, K. Lai, and M. Baker. Mitigating router misbehavior in mobile ad-hoc networks. In *MobiCom*, Sept. 2000.

[30] J. McClave and F. Dietrich. *Statistics*. Macmillan Publishing Company, 6th edition, 1994.

[31] S. Narayanaswamy, V. Kawadia, R. Sreenivas, and P. R. Kumar. Power control in ad-hoc networks: Theory, architecture, algorithm and implementation of the COMPOW protocol. In *European Wireless Conference*, Feb. 2002.

[32] T.-W. Ngan, D. Wallach, and P. Druschel. Enforcing fair sharing of peer-to-peer resources. In *IPTPS*, Feb. 2003.

[33] M. Pritchard. How to hurt the hackers: The scoop on Internet cheating and how you can combat it. http://www.gamasutra.com/features/20000724/pritchard_pfv.htm, July 2000.

[34] B. Raghavan and A. C. Snoeren. Priority forwarding in ad hoc networks with self-interested parties. In *IPTPS*, June 2003.

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

[36] S. Saroiu, K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Multimedia Computing and Networking*, Jan. 2002.

[37] S. Shenker. Making greed work in networks: a game-theoretic analysis of switch service disciplines. *IEEE/ACM Transactions on Networking*, 3(6):819–831, Dec. 2003.

[38] N. Spring, R. Mahajan, and T. Anderson. Quantifying the causes of path inflation. In *ACM SIGCOMM*, Aug. 2003.

[39] V. Srinivasan, P. Nuggehalli, C. F. Chiasserini, and R. R. Rao. Cooperation in wireless ad hoc networks. In *IEEE INFOCOM*, Mar. 2003.

[40] F. Stajano and R. Anderson. The cocaine auction protocol: On the power of anonymous broadcast. In *Int'l Workshop on Information Hiding*, Sept. 1999.

[41] Q. Sun and H. Garcia-Molina. Slic: A selfish link-based incentive mechanism for unstructured peer-to-peer networks. In *24th International Conference on Distributed Computing Systems*, Mar. 2004.

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

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

[44] S. Zhong, Y. Yang, and J. Chen. Sprite: A simple, cheat-proof, credit-based system for mobile ad hoc networks. In *IEEE INFOCOM*, Mar. 2003.

Notes

[1]Interestingly, "TCP accelerators" have been a concern but have not become pervasive because the bottleneck is usually close to the host, implying that there is little to be gained by deviating from the protocol.

[2]In theory, the cheater can pick an arbitrary signal strength range rather than limiting itself to the top end. But our measurements show that the degree of overlap among neighbors in the middle and bottom part of the range would preclude this behavior. Additionally, better signal strength roughly translates to better connectivity, providing an incentive to pick such neighbors.

A Catch Fail-safes

We briefly consider each step of Catch in light of possible, intentional violations by a free-rider. Our goal is to show that a free-rider cannot defeat the protocol by manipulating messages in unanticipated ways.

Epoch-Start Each node must periodically send EpochStart messages or it is deemed uncooperative by its neighbors and is ignored.

Packet Forwarding and Accounting The testee can drop some or all of the challenges. However, because the challenges are anonymous it: i) cannot selectively inflate the loss rate on some of the links and ii) has to waste its own resources if it chooses to uniformly inflate the loss rate on all links. (Section 3.1)

Anonymous Neighbor Verification Open (ANV1) The testee can drop some fraction of the ANV1 messages. However, this will be detected in a reasonably short time because of anonymity. (Section 3.2)

Tester Information Exchange The testee is unable to interfere with the exchange because it relies on all the testers to release their tokens.

Epoch Evaluation and ANV Close (ANV2) It is in the testee's interest to forward these messages since they are required for it to pass the epoch evaluation.

Isolation Decision Testers drop the free-rider's data packets to isolate it. To prevent this punishment from being circumvented, we require that some unforgeable notion of identity transmitted with data packets.

Deliberate False Accusations A different style of attack is for a *tester* to falsely accuse a cooperative testee and cause it to be isolated. The tester is then no longer required to relay packets for this testee. To discourage this, a cooperative testee retaliates by isolating its accuser, or all of its neighbors, if the identity of the accuser is unknown, i.e., mutually-assured-destruction.

Dropping specific data packets A free-rider can use application-level knowledge to throttle data flow if encryption is not used. For instance, it could selectively drop TCP SYN packets at a higher rate to curb data packet generation. We can detect such behavior by looking for statistical differences in the forwarding rate of such special packets.

Blocking control packets Another possibility is for a node to target specific protocol packets sent by other nodes by interfering with their transmission. This is not plausible because we send protocol packets at randomized times.

Reducing transmission power A free-rider can reduce its relaying responsibilities by reducing its transmission power. This requires the node to be topologically well-placed such that there exists a power level at which it has good connectivity to one other node and almost no connectivity to others. Catch does not counter this strategy, as we view power management to be a legitimate strategy for minimizing co-channel interference.