Backbone Construction in Selfish Wireless Networks

Seungjoon Lee
AT&T Labs - Research
slee@research.att.com

Dave Levin
University of Maryland
dml@cs.umd.edu

Vijay Gopalakrishnan
AT&T Labs - Research
gvijay@research.att.com

Bobby Bhattacharjee
University of Maryland
bobby@cs.umd.edu

ABSTRACT

We present a protocol to construct routing backbones in wireless networks composed of selfish participants. Backbones are inherently cooperative, so constructing them in selfish environments is particularly difficult; participants want a backbone to exist (so others relay their packets) but do not want to join the backbone (so they do not have to relay packets for others).

We model the wireless backbone as a public good and use impatience as an incentive for cooperation. To determine if and when to donate to this public good, each participant calculates how patient it should be in obtaining the public good. We quantify patience using the Volunteer's Timing Dilemma (VTD), which we extend to general multihop network settings. Using our generalized VTD analysis, each node individually computes as its dominant strategy the amount of time to wait before joining the backbone. We evaluate our protocol using both simulations and an implementation. Our results show that, even though participants in our system deliberately wait before volunteering, a backbone is formed quickly. Further, the quality of the backbone (such as the size and resulting network lifetime) is comparable to that of existing backbone protocols that assume altruistic behavior.

Categories and Subject Descriptors: C.2.2 [Computer-Communication Networks]: Network Protocols; C.4 [Performance of Systems]

General Terms: Algorithms, Experimentation

Keywords: Wireless backbone, incentives, selfish network, public good, volunteer's dilemma

1. INTRODUCTION

In a multihop wireless network, participants relay packets for others in order to achieve end-to-end connectivity. The benefits of a *backbone* in these networks are well documented [2, 10, 11, 16, 24]. For example, nodes not in the backbone can go into sleep mode and save energy while backbone nodes stay awake and ensure end-to-end connectivity. Prior work on backbone construction, however, has assumed that nodes in the network are inherently cooperative. Other schemes for forwarding and routing mechanisms [3, 26, 27] consider selfish wireless nodes, but rely on external mechanisms such as secure hardware or a central bank to enforce cooperation.

The main contribution of this paper is to show that it is feasible to construct an efficient, power-saving, routing backbone without assuming altruism or resorting to external mechanisms. We consider a network in which participants have data to send (or receive) but are selfish, i.e., they are not inclined to relay packets for others. Thus, nodes in our system want to achieve two conflicting objectives—they do *not* want to join the backbone (since they do not want to relay packets for others), but *do* want a backbone to exist (since they want their own packets to be forwarded). Using tools from game theory, we develop a protocol to construct an efficient routing backbone in selfish environments.

We model the problem of building a backbone as that of creating a *public good*: a commodity from which all nodes benefit. We apply a well-known game-theoretic model called the *Volunteer's Dilemma* [6, 13, 25]. Each participant in the network needs *some* of the nodes to volunteer to provide the public good, but no one wants to be one of the volunteers. The base Volunteer's Dilemma model, however, assumes that all nodes in the network can talk to each other (i.e., a complete graph). We extend this base model to work in any general multihop network. We derive a formula that nodes use to compute the amount of time to *wait* before they volunteer to join the backbone. Our formula takes into account the node's capacity (e.g., remaining battery) and its local neighborhood to compute this waiting time. We show that volunteering at this computed time is the *dominant strategy* for each node; a strategy is dominant if it yields greater utility than any alternative strategy, regardless of what the other players do.

We evaluate the protocol through an implementation (we believe this is the first implementation of a wireless backbone protocol) and extensive simulations over ns-2, and show that the resulting protocol retains the goodness of cooperative backbone construction algorithms. Our results indicate that the protocol builds a small backbone quickly, and nodes with higher battery life are preferentially added to the backbone, thereby saving energy of low battery nodes.

The rest of the paper is organized as follows. We describe our model and assumptions in Section 2. We review the

SIGMETRICS'07, June 12–16, 2007, San Diego, California, USA.

Volunteer's Timing Dilemma and present our generalization in Section 3. Section 4 presents our backbone construction protocol. We present results from simulations in Section 5 and from the implementation in Section 6. We discuss node misbehavior in Section 7, review related work in Section 8 and conclude in Section 9.

2. MODEL AND ASSUMPTIONS

We model the network as a graph $G = (V, E)$, where V is the set of nodes in the network and $(u, v) \in E$ implies that v is within u's transmission range. We make the standard assumption (e.g., as required by the 802.11 MAC) that G is undirected, i.e., each link in the wireless network is bi-directional. We also assume that whenever a node u sends a packet, it is received by each node in its 1-hop neighborhood $\mathcal{N}^1(u)$ as well as the packet's intended recipient. (We denote by $\mathcal{N}^k(u)$ the set of nodes within k hops from u as determined by G, *not* in the backbone.) We use this assumption to detect whether neighbors are faithfully following protocol mechanisms as in [20, 21]. In our protocol, each node uses information about its two-hop neighborhood. We construct a view of two-hop neighborhood for each node by using techniques described in Catch [20]. Like Catch, we assume that each node has a unique ID and that MAC-level authentication is used to prevent Sybil [14] attacks, but we note that we only use these assumptions to obtain each node's two-hop neighborhood information.

Node Utility. We assume the primary concern of each node in our model is to ensure that its connections have high goodput. Since we do not consider any external incentives (e.g., cash), if a node knows that it will not be sending or receiving packets for a significant amount of time, we cannot motivate the node to route and forward packets on behalf of other nodes. In this paper, we assume that connections are made between arbitrary source-destination pairs unpredictably.[1] This is common knowledge among the participants (i.e., all nodes in the system know this, and they know others know this, *ad infinitum*), but nodes do not have any further knowledge about traffic patterns *a priori*.

In this paper, instead of defining a specific utility function, we use the following preference relation. For each node v, v first attempts to maximize its goodput. If there are multiple ways v can achieve equal goodput, then v chooses the strategy that minimizes the energy consumption. Since we assume that nodes do not know when and from which node they will receive messages, ensuring constant *global* end-to-end connectivity is of nodes' primary concern to achieve maximum utility.

Our protocol is intended for deployment when nodes are selfish and can set local policy. This will occur only when no single administrative entity mandates policy on the entire network, for example, in an ad hoc network or a large community rooftop network [1]. However, a *malicious* node could jam the channel and disconnect nodes in a local neighborhood, without regard to its own communication [7]. This attack is inherent in all wireless networks, and is beyond the scope of this paper. When nodes are selfish, it is possible that some of them collude to increase their utility. In this paper, we focus on a simple scenario where nodes are selfish, but do not collude.

[1] Alternately, we only consider nodes that are interested in sending and receiving messages, while other nodes in the network do not participate in the game. As explained in Section 4, these non-participating nodes cannot use the backbone until they play the game in the future.

3. BACKBONE FORMATION: THEORY

In this section, we develop the theory for our backbone formation protocol. We begin with an overview of the well-known Volunteer's Timing Dilemma (VTD) [6] and then describe how we extend VTD to a generalized wireless setting. We describe how to apply the generalized VTD to form a connected backbone in Section 4.

3.1 Background: The Volunteer's Dilemma

Consider the following social dilemma: a group of rational individuals want a *single* person from the group to volunteer to offer some service. This service expends some of the volunteer's resources, but *all* the individuals, including the volunteer, benefit from the service if it is provided. In other words, this service is a *public good*. Without loss of generality, let us assume that each node, $i = 1, .., N$, derives 1 unit of benefit from the existence of this public good and that it costs node i $c_i \in [0, 1]$ to provide the service. Further, the *distribution* $\mathcal{F}$ of these costs is public knowledge, but the cost to any individual node is private (i.e., node i knows how all costs are distributed, but only i knows the precise value of c_i). Although the derivations in this paper generalize to arbitrary distributions, we assume for simplicity that $\mathcal{F}$ is uniformly distributed in the rest of this paper. (In our simulations in Section 5, we experiment with cases when the actual cost distribution is different from the assumed distribution.)

Diekmann [13] presents this formally as a one-shot game called the *Volunteer's Dilemma* (VOD). Each node has two possible strategies it may play: volunteer or free-ride. Player v's utility is:

$$U_v \stackrel{\text{def}}{=} \begin{cases} 1 - c_v & \text{if } v \text{ volunteers} \\ 1 & \text{if someone, but not } v, \text{ volunteers} \\ 0 & \text{if no one volunteers} \end{cases}$$

That is, if at least one node volunteers, everyone obtains the public good and receives utility 1, but each node i that volunteers must pay c_i. If no one volunteers, the public good is not available and no one gains any benefit.

Bliss and Nalebuff [6] consider a slightly different scenario, often called the *Volunteer's Timing Dilemma* (VTD) [25]. In their model, each player's strategy is no longer to "volunteer or not," but rather a time $T \geq 0$ that denotes "when to volunteer." If no one volunteers until time t, then the public good is not available until then. To capture the loss in utility from waiting, each player's utility decreases by the standard exponential discount factor (e^{-t} over time t), giving player v utility $e^{-t}U_v$. As in the VOD, cost is private information but the distribution $\mathcal{F}$ is common knowledge. Bliss and Nalebuff derive $T(n, c_i)$, the optimal time for node i to volunteer by maximizing i's expected utility given its cost, c_i, the distribution of costs, and the total number of players, n:

$$T(n, c_i) = (n - 1) \cdot \left(\frac{c_i}{1 - c_i} + \ln(1 - c_i) \right) \qquad (1)$$

This derivation has several nice properties. First, since $T(n, c_i)$ is increasing on c_i, when all players are rational,

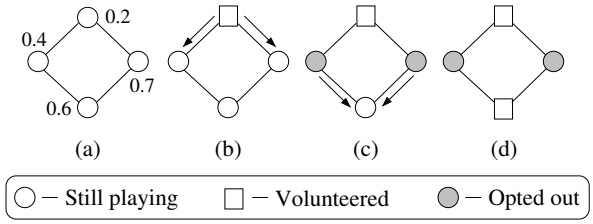

Figure 1: An example GVTD game run on (a) a sample graph. (b) The top-most node volunteers and notifies its neighbors who (c) opt out and inform their neighbor, who then (d) volunteers immediately.

the node with lowest c_i (or highest capacity) is the one to volunteer. Second, as n increases, the optimal volunteering time for each node increases, while the system-wide utility also increases. Last, since T is found by maximizing $e^{-t}U_v$ for each v, T defines a dominant strategy for all players.

3.2 Generalized VTD

Both Diekmann's and Bliss and Nalebuff's models assume that all players can observe and benefit from any volunteer. In multihop networks, however, this assumption does not hold; each node needs a volunteer within its one-hop neighborhood, and therefore does not directly benefit from a volunteer two or more hops away. In this section, we introduce our generalization that we call the Generalized Volunteer's Timing Dilemma (GVTD), where an input to the game is an arbitrary, undirected graph G. Note that the original VTD game is a special case where G is a complete graph. We now present a derivation of the optimal time that each node should wait before volunteering in a general graph G, and show that the final set of volunteers constitutes a maximal independent set of G.

3.2.1 Optimal Waiting Time in GVTD

In GVTD, nodes can continue playing the game even after others announce their action and leave the game (Figure 1). A node v can volunteer if it has waited long enough and none of its neighbors have yet volunteered. After v volunteers, its neighbors can receive the public good from v and therefore stop playing (*opt out*) of the game. However, nodes more than one hop away from v keep playing the game until they or one of their neighbors volunteer.

Similar to VTD, each node v playing GVTD first calculates its optimal waiting time. To do so in VTD requires only v's volunteering cost, but the waiting time in GVTD depends on both the cost and the network topology. An accurate prediction of the cost incurred by volunteering would require v to have an estimate on how much traffic its neighbors wish to send, which is often difficult to obtain. In our implementation and simulation, we let c_v be $(1 - r_v)$ where r_v is v's remaining battery; were the lifetime of the backbone not the main concern, one could envision using, say, bandwidth or latency in determining the cost, instead.

Although a node needs to know the global topology to calculate the optimal time, obtaining such information is typically costly. In this paper, when a node calculates its waiting time before volunteering, it uses its *two-hop neighborhood* information, which is obtained using techniques suggested in

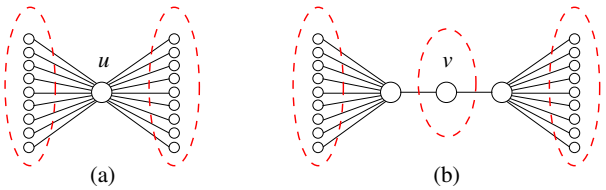

Figure 2: Dashed ovals represent likely volunteers. (a) A large one-hop neighborhood reduces u's probability of volunteering, whereas (b) a large two-hop neighborhood has the opposite effect.

Catch [20]. However, GVTD generalizes to the global optimal if a consistent view of the entire topology is available. Let $\mathcal{R}^1(v)$ be the one-hop neighbors of v who have not opted out (i.e., v's neighbors remaining in the game). For a neighbor u of v, let $n_{u\setminus v}$ denote the number of one-hop neighbors of u who are *not* one-hop neighbors of v. Then, we prove that v can determine how long it should wait before volunteering using:

$$T_v(c) \approx \int_{x=0}^{c} \sum_{u \in \mathcal{R}^1(v)} \frac{(n_{u\setminus v}+1)\; x(1-x)^{n_{u\setminus v}-1}}{n_{u\setminus v} + (1-x)^{n_{u\setminus v}+1}} \quad \mathrm{d}x \quad (2)$$

We present the derivation in the appendix. Note that in the complete graph case, $n_{u\setminus v} = 0$, $\mathcal{R}^1(v)$ is either $\mathcal{N}^1(v) = V \setminus v$ or $\emptyset$, and $|\mathcal{N}^1(v)| = (|V| - 1), \forall u, v \in V$. Then, denoting $|V| = n$, we have $T_v(c) \approx \int_0^c (n-1)\frac{x}{(1-x)^2}\,\mathrm{d}x$, which reduces to Eq. 1 for the original VTD analysis.

Observe that T_v reflects the amount of time to wait to volunteer since the beginning of the game. In GVTD, since $\mathcal{R}^1(v)$ changes over time, T_v also changes. Hence, it is possible that T_v becomes less than the current time elapsed in the game, in which case v volunteers immediately. In Figure 1(d), the bottom-most node recalculates T with no remaining neighbors ($\mathcal{R}^1 = \emptyset$) and therefore volunteers immediately. Note also that T_v does not take into account system-wide connectivity; this fact allows for a purely distributed protocol which we present in the following section.

3.2.2 GVTD Solution Properties

Our derivation in Equation 2 yields many of the same properties of the original VTD. First, as in the model proposed by Bliss and Nalebuff, T_v in the GVTD is increasing in c_v; this ensures that when all other factors are equal, nodes with lower cost (or high capacity) volunteer earlier. Next, T_v is increasing in $\left|\mathcal{N}^1(v)\right|$; this implies that each additional one-hop neighbor is another candidate to allow v to opt out rather than volunteer itself. For example, in Figure 2(a), node u is unlikely to volunteer, as it has many one-hop neighbors who may do so earlier.

New to the GVTD is the notion of a non-trivial $\mathcal{N}^2(v)$; as this grows, T_v decreases. To see this, note that each additional two-hop neighbor is another candidate for (at least) one of v's one-hop neighbors to opt out. In Figure 2(b), node v is likely to volunteer, since each of its one-hop neighbors is likely to opt out.

GVTD does not always result in the nodes with the lowest cost volunteering. Suppose in Figure 2(a) that node u has cost 0.1 and every other node has cost 0.99. Though node u has the lowest cost, all of its neighbors have degree 1, hence they will all have significantly larger probability of volun-

teering and therefore smaller values of T. In this example, $T_u(c_u) = 0.1$ and, for each neighbor v of u, $T_v(c_v) = 0.003$. Such an effect is a necessary outcome of this game. This is due to both private information and the graph's topological constraints. Since each node u does not know any other nodes' cost to volunteer, it can at best estimate the probability that its neighbors will volunteer before it does.

GVTD Yields a Maximal Independent Set. Recall that an independent set S of $G = (V, E)$ is a subset of V such that no two vertices in S correspond to an edge in E. S is a maximal independent set if no proper superset of S is an independent set. Recall also that for a *dominating set* D, each node in V either is in D or has a neighbor in D.

THEOREM 1. *Given an input graph $G = (V, E)$, when the GVTD game ends, the set of volunteers constitutes a maximal independent set of V.*

PROOF. Let $U \subseteq V$ denote the set of nodes that volunteered at the completion of the VTD game. For each node $v \in V$, we have, exactly one of the following: $v \in U$ or $\exists \alpha \in \mathcal{N}^1(v) \cap U$. By definition, U is an independent *dominating* set, and is therefore a maximal independent set. □

In the unit-disk graph model [16], which is a simple yet popular model for wireless networks, a maximal independent set is a constant-factor approximation of a minimum dominating set. Finding a minimum-sized dominating set is a well-known NP-hard problem [17], and Theorem 1 proves that in the unit-disk graph model, the resulting dominating set after a GVTD game is essentially as small as possible. In fact, our protocol ensures our resulting *connected* backbone is also a constant-factor approximation to a minimum connected dominating set (see Section 4). In Figure 2(a) and (b), the dashed ovals show a maximal independent set for each of two graphs.

These GVTD properties are now sufficient background to construct a protocol for backbone formation in selfish networks, which we describe next.

4. BACKBONE FORMATION: PROTOCOL

Like many existing backbone construction schemes [2, 16, 24], our backbone construction protocol consists of two logical steps: leader selection and the connection of the leaders. In the first phase, nodes play the GVTD game. Based on the information about the cost distribution (Section 3) and its two-hop neighborhood, each node independently computes its optimal waiting time before volunteering. When there is no volunteer neighbor for a long (enough) time, it volunteers as a *leader* to speed up the backbone construction, and thus minimizes loss of its own messages. In the second phase, we choose *bridge nodes* to connect the leaders and obtain a connected backbone (specifically, a connected dominating set). In this section, we describe the backbone formation protocol using the IEEE 802.11 terminology; however, the protocol is not particularly tied to 802.11.

4.1 Leader Selection Protocol

Initially, we assume each node is in sleep mode. Nodes wake up periodically (*e.g.*, as in IEEE 802.11) and exchange their neighbor information. This information includes IDs of the transmitting node and all its neighbors. Each node also checks if any neighbor has volunteered. If node v does not observe any volunteers for a period longer than its optimal waiting time, $T_v(c_v)$, it volunteers as a leader for a pre-defined service duration, and broadcasts a LEADERDECL message to its one-hop neighbors. Upon receipt of this message, all of v's neighbors know that they have a volunteer; they opt out of the game and include the leader information in subsequent periodic messages. Our leader selection protocol is *repeated* when the service duration expires, which we further elaborate on in Section 4.3.

Incentives for Truthfulness. Recall from Section 3 that v will volunteer sooner if its one-hop neighborhood is sparser. Thus, it may appear that v's neighbor u would choose not to broadcast its identity, so that v would not count u as a neighbor and would therefore volunteer earlier. However, if v becomes a leader, v will not regard u as a neighbor and will not provide the backbone service to u. Hence, "hiding" is of no use to u.

Another way for u to shorten v's waiting time is to pretend u has many neighbors by including fake neighbors in periodic messages. This is, in effect, a Sybil attack [14]. We consider this an orthogonal problem, and direct the reader to Newsome *et al.* [22] who detect and defend against Sybil attacks in a wireless setting.

Estimating Cost Distribution. As discussed in Section 3, we assume that each node knows its own cost and also the distribution of costs of other nodes. In practice, various factors determine the cost, including remaining battery power, the degree of desire for communication and altruistic tendency. A network composed of many nodes with cost close to 1 corresponds to the scenario where many selfish nodes are reluctant to volunteer. In contrast, if there are many low-cost nodes, then the network is more altruistic and the backbone construction is faster. One way is to learn relative willingness of neighbors over time and use them as sample points to infer the distribution. In our results, we quantify the convergence time and quality of the backbone when the estimated cost distribution is different from the actual. However, there exists little study on altruistic behavior in real multihop wireless networks, and inferring the overall cost distribution is an open question.

4.2 Connecting the Leaders

Observe that, after leader selection, no two leaders will be adjacent. The second phase of our backbone construction consists of choosing nodes between leaders to act as *bridge nodes* to form a connected backbone.

Since the final leader set is a dominating set (Theorem 1), each leader will be no more than three hops away from at least one other leader [17]. It therefore suffices for each leader node to learn about other leaders that are reachable within three hops (i.e., via a path through at most two non-leaders). To accomplish this, leader node u broadcasts to its neighbors a message indicating that it has volunteered. u's neighbors then forward this message to their one-hop neighbors. Each neighbor v of u then requires its neighbors, $\mathcal{N}^1(v) \setminus \mathcal{N}^1(u)$, to forward this message to their one-hop neighbors. We discuss v's incentive to forward this message in more detail later in this section.

After these messages are propagated, each node v knows of all leaders (denoted by L_v) in its 3-hop neighborhood; v also knows of all paths (of length at most 3) from v to each node

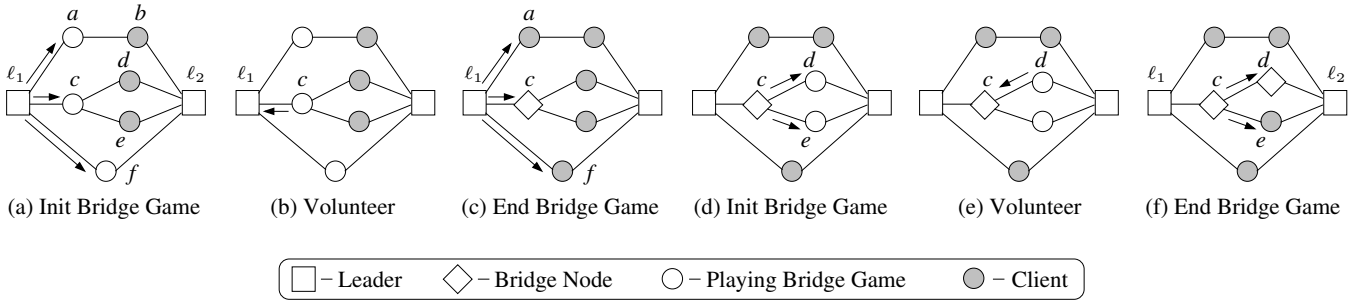

Figure 3: Bridge node selection between two leaders, ℓ_1 and ℓ_2, consists of playing a bridge game at each of the (at most two) hops on a path from ℓ_1 to ℓ_2. The winners of the respective games are (c) node c and (f) node d, resulting in the path (ℓ_1, c, d, ℓ_2).

in L_v. This information can be used to connect the leaders in many different ways, such that different metrics (e.g., backbone size [16, 17], or minimum cost [24]) are optimized. In our protocol, we consider the simplest case where each leader connects to *all* other leaders that are in its 3-hop neighborhood. Such a backbone has some nice properties: the increase in hop count for any path is within a constant factor compared to the base case without a backbone, and the backbone size is a constant-factor approximation of a minimum backbone [2]. Still, the backbone is in general larger than strictly necessary, and the overall energy saving is accordingly smaller. However, as we discuss further in the following section, we may need to detour around some misbehaving nodes, in which case we benefit from redundant paths due to larger backbones. Obtaining backbones that are small yet allow for resilience against misbehaving nodes is an interesting subject of future work.

Bridge Node Selection. For a leader to connect to other leaders, it must select a set of *bridge nodes* to forward packets between leaders. When leader ℓ_1 wants to connect to leader $\ell_2 \in \mathcal{N}^3(\ell_1)$, it uses the following bridge node selection process. (To break symmetry, only the leader with the smaller ID initiates the bridge node selection process.) First, using its knowledge of its 3-hop neighborhood, ℓ_1 determines all of the available paths of length at most 3 from itself to ℓ_2. Let $\mathcal{B}(\ell_1 \to \ell_2)$ denote the set of one-hop neighbors of ℓ_1 who are on at least one of these paths. In Figure 3, $\mathcal{B}(\ell_1 \to \ell_2) = \{a, c, f\}$. Then, ℓ_1 sends a PlayBridgeGame message to $\mathcal{B}(\ell_1 \to \ell_2)$.

Algorithm 1 Play Bridge Game$(\ell_1, \ell_2, \mathcal{B})$

Called by u when it receives a PlayBridgeGame message from prev_hop to connect ℓ_1 to ℓ_2. $\mathcal{B}$ is the list of potential volunteers.

```
1: if u ∉ B
2:    return
3: t ← Calculate-Complete-VTD-Time(c_u, |B|)
4: Wait for time t
5: if after waiting, prev_hop has not announced a volunteer
6:    Send BridgeVolunteer message to prev_hop
7:    if prev_hop replies with a BridgeAck
8:       if ℓ2 ∉ N^1(u)
9:          B' ← {v : v ∈ N^1(u) ∧ ℓ2 ∈ N^1(v)}
10:         Broadcast PlayBridgeGame(ℓ1, ℓ2, B')
```

We provide pseudocode for our bridge selection protocol in Algorithm 1. To summarize, each hop on the path from ℓ_1 to ℓ_2 is obtained by applying VTD. Since each node that receives a PlayBridgeGame message has the complete information about $\mathcal{B}$, nodes in $\mathcal{B}$ play the standard, complete graph VTD game using Eq. 1. The first game is played by the nodes in $\mathcal{B}(\ell_1 \to \ell_2)$. Each node playing the game stops either when it volunteers and informs the previous hop on the path, or when the previous hop broadcasts an acknowledgment to the first volunteer. In Figure 3, ℓ_1 initiates the first game (a); node c wins and becomes a bridge node when it receives an acknowledgment from ℓ_1 (b)-(c). Note that nodes a and f stop playing the game once they hear the acknowledgment from ℓ_1.

After the winner of the first game joins the backbone, it either informs ℓ_2 of the path (if it is connected to ℓ_2), or else initiates another VTD game to obtain the next hop. This second game is played by the nodes who are both one-hop neighbors of the new bridge node and ℓ_2. In Figure 3, c is not connected to ℓ_2 and initiates a new game (d), which node d wins (e)-(f), resulting in the path (ℓ_1, c, d, ℓ_2). Note that, since ℓ_1 tries to connect to leaders within three hop distance, this process will require at most two such games.

Bridge Selection without VTD Game. The bridge selection algorithm described above incurs some overhead due to the waiting times and control message exchanges. An alternate scheme is to forgo the bridge nodes' VTD games and to just have ℓ_1 designate bridge nodes. If a designated node refuses the request, then ℓ_1 in turn can refuse to provide the backbone service to the node. With this alternate scheme, the bridge selection is immediate, but the backbone may include high-cost nodes. In our simulations, we experiment with the full VTD-based bridge selection algorithm; in our implementation, leaders are chosen using GVTD, but bridges are assigned by leaders without undertaking the VTD game.

Workloads of Leaders and Bridge Nodes. Although both leaders and bridges stay awake and relay messages for other nodes, the key difference in terms of workload is that leaders accept clients and buffer their packets (while the clients sleep). There are also several backbone construction schemes that do not strictly differentiate these two types [10, 11]. Designing such a backbone scheme based on the GVTD analysis is an interesting future research topic.

4.3 Repeated Game

Since battery levels decrease over time, the backbone must be reconstructed over time to ensure that it consists of high-capacity nodes. We next describe the specific policies necessary to repeatedly play the backbone games. Node failures can be treated the same as node departures, and are handled by this repeated game construct.

When a node becomes a leader, it implicitly enters a contract with its neighbors to perform the role of a leader for a system-defined service duration. A leader ℓ could attempt to misbehave by ceasing to forward packets or by not connecting to the other leaders within its 3-hop neighborhood. We discuss how to address this and other forms of misbehavior in Section 7.

When a leader ℓ's service duration has passed, it may leave the backbone, or "unvolunteer." To do so, the exiting leader broadcasts an UNVOLUNTEER packet and immediately begins playing GVTD. Each of ℓ's neighbors, $i \in \mathcal{N}^1(\ell)$, begins playing the GVTD game or, if there is some other leader in $\mathcal{N}^1(i)$, becomes a client of that node instead. The rest of the protocol follows as in Sections 4.1; someone in ℓ's 1-hop neighborhood will volunteer, and the rest become clients or bridge nodes.

To ensure that bridge nodes do not leave their associated leaders when a neighbor volunteers, we use a similar implicit-contract-based policy. When a node volunteers to be a bridge node to connect its leader ℓ to a set of leaders $\{\ell_1, \ell_2, \ldots\}$, it is required to continue being a bridge node until either its leader unvolunteers or until each ℓ_i unvolunteers.

The Need for Incentives. Consider what incentive a node has to maintain its role in the backbone, or to forward correct neighbor information. Our protocol does not necessarily provide incentive to perform tasks such as these, and hence a further incentive mechanism is required. Such a mechanism is orthogonal to our protocol, and could even be out-of-band, like cash. In Section 7, we demonstrate that the same punishment mechanism can be used to keep leaders and bridge nodes in the backbone, as well as to ensure that nodes transmit correct information to their neighbors.

5. SIMULATION RESULTS

In this section, we present results from ns-2 simulations of our backbone construction protocol. First, we study the performance when all nodes in the system are rational, that is, no participants deviate from the protocol. Then, we investigate the system's performance when nodes' prior assumptions are false. We also analyze the price of irrationality by allowing some of the nodes in the system to deviate by refusing to be either a leader or a bridge node. Finally, we compare the performance of our backbone algorithm with SPAN, a cooperative backbone construction scheme. We conducted our studies on three different topologies, each generated from an urban setting in Portland, OR, modeling transceivers placed along the roadways, for 130, 227, and 306 nodes [4].

All Peers Rational and Well-Informed. In this set of experiments, all peers follow the bridge construction protocol as detailed in Sections 3 and 4 (i.e., they are rational), and each node knows that the distribution of costs is uniform

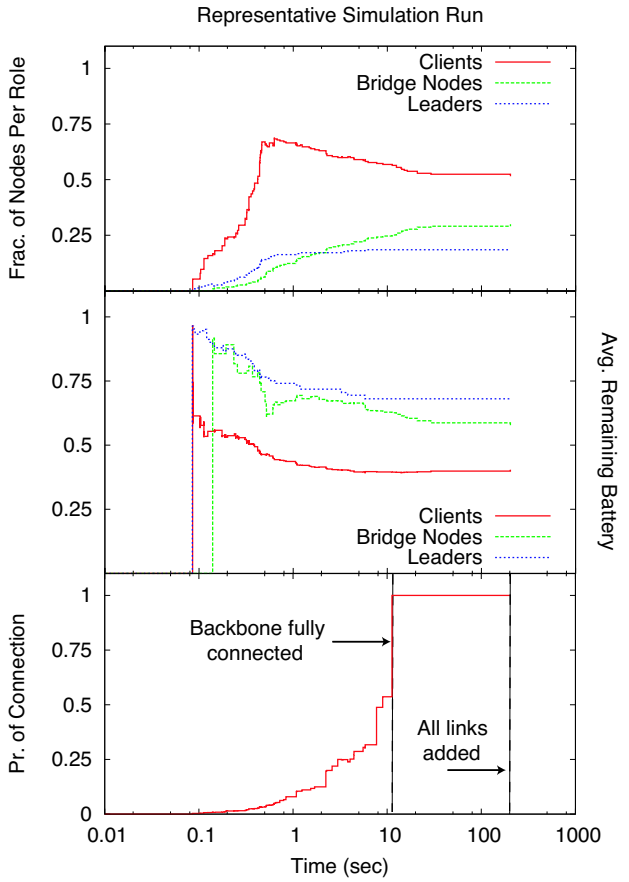

Figure 4: Representative backbone properties over time, $N = 227$.

(i.e., they are well-informed). We present a sample run of this scenario on the 227 node topology in Figure 4, and averages of 10 runs for different topologies in Figure 5.

As shown in the top plot of Figure 4, the majority of nodes are clients, and hence more than half of the network can enter sleep mode. In fact, *the greater the amount of overhead required to perform a role, the fewer number of nodes that must perform that role*; there are fewer leaders than bridge nodes, and fewer bridge nodes than clients. This yields a longer network lifetime, as the fewest number of nodes are assigned to the most costly roles. This trend is clear in other cases as shown in Figure 5 as well.

One effective way of maximizing network lifetime is to have high-battery nodes in the backbone, while putting low-battery nodes to sleep. As shown in the middle plot of Fig. 4, our system chooses the nodes in precisely this fashion. As time progresses, higher-capacity nodes are chosen first. Only as the need permits are lower-capacity nodes eventually added to the backbone. When there are no better choices for bridge nodes between a pair of leaders, spikes in bridge node costs occur, such as around time 0.5.

Although our protocol requires nodes to wait for their GVTD timer to expire before volunteering, it forms connected backbones quickly. This is captured in the bottom plot of Fig. 4, in which we show the probability of a random source-destination pair being able to communicate over the backbone. A connected backbone is formed very

N	Backbone Completion Time (s)	
	Connected	All Links
130	14.7 (13.0)	231 (230)
227	23.9 (14.4)	231 (202)
306	34.7 (22.1)	379 (234)

Table 1: Average backbone construction time when all peers know the correct distribution of costs (uniform). The values in parentheses denote standard deviations.

N	Avg. Rem. Batt.			Completion Time (s)		Cvg. of Max
	Ldr	Brg	Cli	Connected	All Links	Conn. Comp
130	.397	.215	.064	873 (414)	1849 (454)	.986 (.0108)
227	.357	.226	.073	1560 (236)	1970 (390)	.908 (.109)
306	.376	.223	.065	1860 (656)	2170 (495)	.985 (.0170)

Table 2: Results when all peers believe that the cost distribution is uniform when it actually is long-tail (majority of nodes have low battery). Due to many low-battery nodes, only 25% of the runs resulted in a fully connected component. The completion times shown are from those runs only. The values in parentheses denote standard deviations.

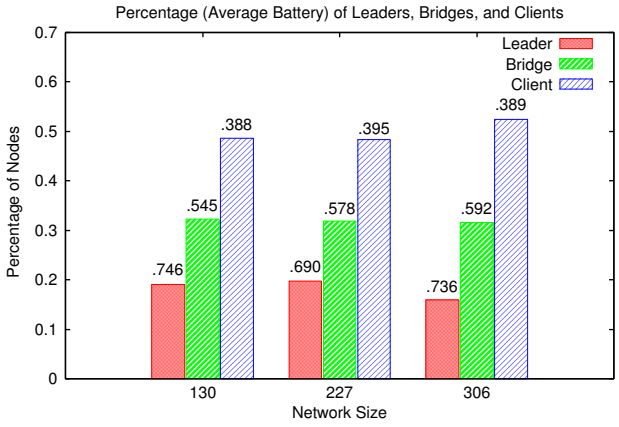

Figure 5: Percentage of leaders, bridges, and clients in different topologies. The value on top of each bar denotes the average remaining battery for each case.

quickly (10 sec in this plot). Since nodes only have information about their local neighborhoods, they continue playing bridge games until every leader is connected to each other leader that is within 3 hops. It can take a considerable amount of time to add all of these redundant links (more than 3 minutes in this case). We show further evidence of this in Table 1. For example, with 306-node networks, it takes around 34 seconds to form a connected backbone, but 379 seconds to add all the links. Though many redundant links are added, the backbone is not adversely affected; as shown in the top of Fig. 4, few nodes are promoted into the backbone once it is connected.

The Effect of Incomplete Information. In the GVTD analysis, we assume that the distribution $\mathcal{F}$ of nodes' costs-to-volunteer is public knowledge. Since this assumption may not hold in practice, we performed experiments to understand the effect of incomplete information on our protocol. In this experiment, each node in the system assumes that the cost distribution $\mathcal{F}$ is uniform, as we did in our analysis, when in reality costs actually follow a different distribution. Since it is unclear what a reasonable distribution of battery values is in practice, we experimented with various distributions. In this section, we present our results using the Pareto distribution; we chose this because it is substantially different from the assumed uniform distribution.

We present our results in Table 2. The quality of the backbone our protocol constructs is very resilient to even vastly inaccurate prior assumptions. For all system sizes in Table 2, the distribution of costs for each role remains the same as when the nodes are well-informed; leaders still have the highest remaining capacity, and so on. The fundamental difference is in the completion time; the average completion times are orders of magnitude longer than the case with uniform distributions (see Table 1). This follows directly from the analysis in Section 3; since most nodes' battery levels are very low, their assumptions of a uniform distribution leads them to believe that their neighbors *must* have greater capacity, and would therefore be willing to volunteer first. The coverage of the maximum connected component, also shown in Table 2, shows that there is a small percentage of nodes who have so little remaining battery that they are willing to wait indefinitely for one of their neighbors to volunteer; if they joined the backbone, they would immediately lose their remaining battery. In 75% of our experiments, these nodes were willing to wait longer than the duration of the simulation (20 min).

Besides the long-tail distribution, we also ran experiments when the actual costs follow a normal distribution. Our findings were similar, and lead us to the following conclusion: *The distribution of costs according to roles remain independent of the accuracy of the information. However, the time to form a fully connected backbone can change significantly.* When nodes' battery levels are lower than what they expect the average to be, they become increasingly patient, and the backbone takes more time to converge (Table 2). Conversely, when nodes have what they perceive to be a higher-than-average battery level, they become more willing to volunteer, and the backbone forms significantly faster.

The Price of Free-Riding. We study the effect of free-riders on the system. By free-riding, we mean that the node refuses to take any role in the backbone. We show that free-riding has an adverse effect on *each* node in the network, including the free-riders, and conclude that free-riding is not a rational strategy.

We experimented with a varying number of nodes acting as free-riders on the 130-node topology. Clearly, if all nodes deviated in such a manner, the network would be completely disconnected and yield no social benefit.

As expected, we see in Figure 6 that the connectivity quickly declines with respect to the number of free-riders in the system. In other words, *rampant free-riding causes system collapse*, so utility-maximizing, rational nodes will have no incentive to free-ride on such a large scale. When only a small percentage of nodes refuse to be either leader or bridge nodes, the backbone can still be connected. In the inset to Figure 6, we focus on the regime with few free-riders (between 0% and 10%), and include only the runs that re-

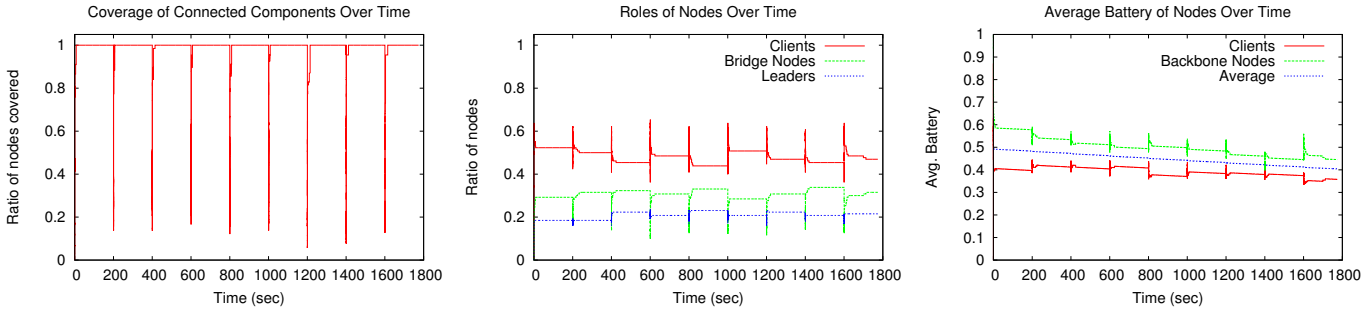

Figure 7: Repeated Game: Connectivity (left), fraction of nodes per role (center), and average remaining battery (right) from a representative run. $N = 130$.

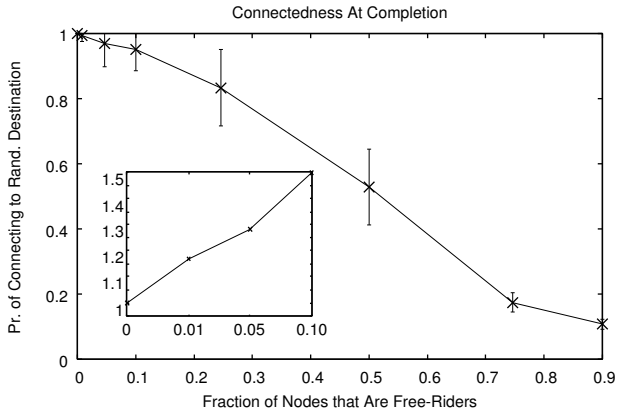

Figure 6: Given an increasing number of nodes who refuse to take part in the backbone, the probability of any node (selfish or not) being able to connect to a destination chosen uniformly at random declines. Even when the network forms a connected backbone, the free-riders delay the process (inset). $N = 130$.

sulted in a connected backbone. On the y-axis, we plot the factor increase in time it takes for a connected backbone to form over the time it takes when all nodes are rational. *Even when it does not fragment the network, free-riding negatively affects* each *node's utility by delaying the backbone from becoming connected.* Hence, a rational node trying to maximize its network connectivity would not free-ride.

Repeated Game. We next investigate how repeatedly playing the backbone games affects the quality of the backbone over time. We ran the repeated version of the game on the 130-node topology, which each node joined at the same time. For simplicity, we set the service duration to 200 seconds in this experiment. Events are not synchronized among nodes; leaders inform their neighbors when they "unvolunteer" from the backbone when their service duration is over.

Our protocol requires the backbone to become disconnected to provide proper incentive for nodes to volunteer (Section 4.3). The connectivity plotted in Figure 7 shows that the backbone recovers extremely quickly, even when all of the leaders in the system unvolunteer within a short time of one another. The downward spikes in connectivity occur when nodes stop being leaders; a new GVTD game begins immediately thereafter. The connectivity drops to as low as 10% at these points, but almost immediately (fractions of a second) returns to 80-90% connectivity. The time to connect the final 10-20% of the network varies among iterations, as different leaders and bridge nodes are chosen. Some, like the iteration at 200s, experience virtually no loss in connectivity, while others, like at 1400s, take several seconds to connect the final 5% of the network.

Throughout the iterations of the repeated game, the fraction of nodes corresponding to each role remained consistent with those in the one-shot game experiments. As shown in the center plot in Figure 7, even as the individual participants change over time, the absolute *number* of participants in this representative run varied little with each iteration of the game. This low variance is to be expected, since our backbone has at most a constant factor more nodes than optimal (Section 3.2).

To investigate whether the backbone always consists of high-capacity nodes, we allow nodes' batteries to decay during the simulations. We differentiate the amount of energy consumption for each operating mode according to the actual measurement in [10]. In Figure 7 (right), we compare the average remaining battery of backbone and non-backbone nodes to the system-wide average. During all repetitions of the game, the average capacity of backbone nodes is consistently greater than the system-wide average. The converse holds for clients; at any point in time, *nodes whose battery levels are less than the system-wide average are less likely to have to further drain their batteries by participating in the backbone.*

The overall quality and lifetime of any backbone is maintained by not just choosing high-capacity nodes, but by alternating which of the high-capacity nodes volunteers over time. This phenomenon occurs naturally within our protocol. Figure 8 shows the positive correlation between nodes' remaining battery and the time they spend in the backbone. These values are averaged over 10 different runs, each with a different set of initial battery levels. There are some low-capacity nodes in the backbone. This is due to private information used in the protocol and topological constraints in the network (e.g., articulation nodes).

Comparison with a Cooperative Scheme. We use SPAN [10] as an example scheme for cooperative backbone construction and compare it with our scheme. We ran the SPAN code provided by its authors on the 306-node network with uniform distribution of battery levels, recording values as soon

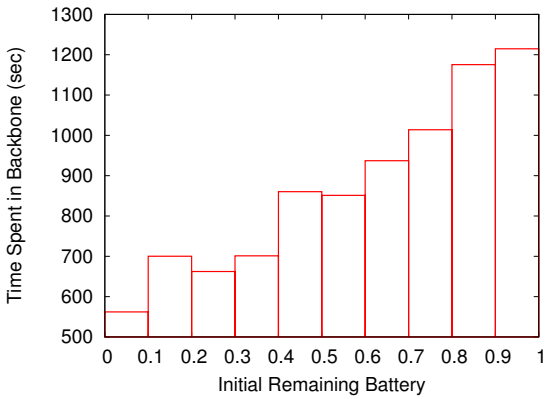

Figure 8: Repeated Game: Average time spent in the backbone versus remaining battery. $N = 130$. **Note that the y-axis starts at 500.**

as SPAN constructed a connected backbone.

On average, SPAN includes 84 of 306 nodes in the backbone: 61 fewer than our scheme (Figure 5). However, our scheme includes more backbone nodes by design so that messages can detour areas with misbehaving nodes using redundant paths (Section 4.2). Interestingly, although our proposed scheme does not assume cooperation, we achieve higher average battery level of backbone nodes than SPAN (0.640 vs. 0.543). This comes from the fact that, in addition to battery levels, SPAN uses randomization when selecting backbone nodes and thus includes many low-battery backbone nodes. As expected, the cooperative nature of SPAN leads to shorter convergence time of 12.9 sec than that of our proposed scheme (34.7 sec). In summary, our results indicate that *our proposed incentives-compatible scheme can achieve similar performance to existing backbone construction schemes that assume cooperation.*

6. IMPLEMENTATION

In order to understand the real-world performance of our backbone protocol, we implemented our scheme and ran experiments over a testbed consisting of laptops equipped with wireless cards. In this section, we first describe the testbed and our implementation before presenting results from our experiments.

Testbed. We performed our experiments using 12 laptops running GNU/Linux (kernel 2.6.11). All the laptops were equipped with an 802.11g wireless card with the Atheros chipset and controlled using the MadWifi driver. These laptops were spread across different rooms on the same floor, as shown in Figure 9.

We implemented the main protocol component in the user level by making appropriate additions to the Click Modular Router from MIT (`http://pdos.csail.mit.edu/click/`). We also implemented driver-level changes to the MadWifi driver in order to support sleep-mode operation. We further describe these changes later in the section.

In our experiments, we measured the end-to-end throughput and latency for different source-destination pairs, sending traffic over the backbone resulting from running our protocol. We measured the end-to-end TCP throughput using `netperf` (`http://www.netperf.org`) and end-to-end latency using `ping`. We let the network stabilize by exchanging control messages (e.g., routing probe messages) between nodes, before the actual transfer of application-level data. We repeated each experiment multiple times and used the average of five different runs in this paper.

Backbone and Routing. The backbone protocol is implemented entirely in the user space as a Click element and uses our modified MadWifi driver to exchange packets. To identify routes and forward packets between source and destination, we use the Click routing element by the Roofnet project (`http://pdos.csail.mit.edu/roofnet/`) that implements the routing protocol proposed by Draves et al. [15]. We modified the routing protocol to reflect the rules imposed by our backbone scheme, so that data packets are routed along the backbone. All packets generated by or destined to clients always go through their leader. When clients are in sleep mode, their leaders buffer their incoming packets until they wake up and request the packets. (We describe more detailed mechanisms later in this section.) Note that bridge nodes can receive packets directly from other backbone nodes in their vicinity (both leaders and bridge nodes).

The backbone layer is responsible for exchanging control messages for backbone construction. Some of the messages are periodic and used to disseminate the network state such as neighborhood information. The backbone layer determines the appropriate waiting time based on the aggregate neighborhood information and the amount of remaining battery. We use the mechanisms provided by ACPI (Advanced Configuration and Power Interface) to retrieve the battery level. The backbone layer also sends out other control messages such as LeaderDecl and executes appropriate device driver commands depending on the node's role (e.g., putting the node to sleep).

In our experiments, we observe that there exist a significant number of high-error and uni-directional links. To achieve efficient communication along the backbone, we should not use those unreliable links to connect backbone nodes. Although the problem of selecting good wireless links for a path is well studied [12, 15], constructing a backbone composed of both high-capacity nodes and high-quality links is an open problem. In this paper, we use a simple threshold-based scheme and ignore a link whose average signal strength is lower than the threshold.

Driver-level Changes. The stock MadWifi driver for the Atheros Chipset supports the usual managed, access-point (AP), and ad-hoc modes. In order for our protocol to work, we need to be able to put devices to sleep, buffer packets at the leader for clients in sleep mode, and communicate directly between neighbors. Unfortunately, *no* single operating mode supports all these operations.

We now describe our extension to the MadWifi driver to implement our protocol. The 802.11 AP mode operation natively supports buffering for *associated* nodes that are asleep. Hence, we force the leader nodes into the AP mode and force the non-leaders (i.e., bridges and clients) into managed mode, with a leader as their associated base station. To ensure that non-leader nodes stay connected with their leader, we disable the default periodic scanning in the driver (that searches for better APs nearby).

In our protocol, clients conserve energy by switching their

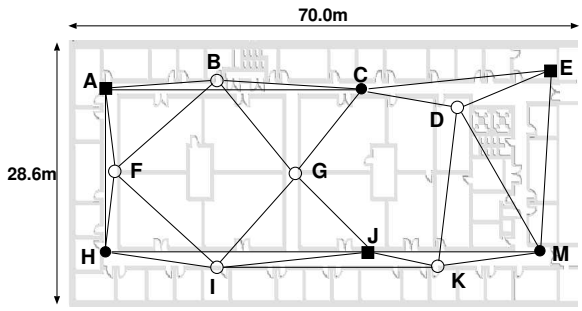

Figure 9: Experiment layout. In our backbone experiment, 50% of the nodes (B, D, F, G, I, K) are in sleep mode. Nodes with square (A, E, J) are leaders, nodes in black dots (C, H, M) are bridges.

radios into the sleep state. However, the stock MadWifi driver does not support switching radios into the sleep state in any of the operating modes. Hence we extended the driver to support sleep-state operation of devices that are in managed mode. Clients enter the sleep state when they have no packets to send or receive. Then, they periodically wake up to receive *beacon* messages (100 ms in our implementation) and check whether the leader has data packets buffered. The node stays awake for the remaining beacon period if it has an incoming message buffered at the leader or an outgoing message in its own queue[2]; otherwise, it goes back to sleep.

With the stock MadWifi driver, APs can talk to nodes associated with it, or with other APs. Nodes in managed mode can communicate only with their APs. Our protocol, however, demands that managed-mode nodes exchange packets with one another. Hence we have extended the driver to permit unrestricted communication between neighbors, regardless of their operating mode.

Finally, in our testbed, the transmission range of any node at low rates (e.g., 1 Mbps) was too large, and we could not obtain interesting topologies. To reduce the effective range, we pinned the MAC-level transmission rate at 11 Mbps for all frames generated by the backbone protocol and application. We can achieve a similar effect by reducing the transmit power, but the driver did not support the functionality.

6.1 Results

In this section, we investigate the effect of using a backbone on overall network performance. We first compare the end-to-end throughput and latency for various source-destination pairs. We then quantify the amount of energy saved by each client.

Effect of the Backbone. In Figure 9, we show the backbone resulting from our protocol. In our experiments, we consider the following flows with and without the backbone. For Flow 1 ($E \rightarrow A$), we use the same shortest path (E-C-A) with and without the backbone, since all the nodes on the path are in the backbone. In this scenario, we expect two performance values (with and without the backbone) to be similar. For Flow 2 ($I \rightarrow M$), the path on top of the backbone (I-J-M) is the same as the case without the backbone, but the source I is a client and in sleep mode. From this scenario, we can observe how the performance changes when we use the same path, but an end node is not in the backbone. For Flow 3 ($E \rightarrow G$), the path on the backbone (E-M-J-G) is different from the path without the backbone (E-C-G), and the destination G is a client. In this scenario, we can observe how much performance degradation we experience with the backbone.

[2]An alternate option is to allow a node in sleep mode to wake up and send a message immediately whenever it has data to send.

	Backbone		No Backbone	
	T'put (Mbps)	Latency (ms)	T'put (Mbps)	Latency (ms)
Flow 1: $E \rightarrow A$	2.36	4.07	2.45	3.64
Flow 2: $I \rightarrow M$	2.42	56.38	2.38	3.93
Flow 3: $E \rightarrow G$	1.28	64.58	2.32	3.62

Table 3: Throughput and latency with and without backbone. Flow 3 follows a longer path over the backbone, resulting in a drop in throughput. With the backbone, Flows 2 and 3 have a sleeping node, which results in larger latency.

In Table 3, we compare average throughput and latency of each flow with and without a backbone. For Flow 1, as expected, both end-to-end bandwidth and latency are largely unaffected by the backbone since both the path and node state (awake or asleep) remain unchanged. Flow 2, on the other hand, experiences higher latency when using the backbone. It is because node I is not in the backbone and is in sleep mode (Figure 9). We still observe similar results for throughput, since node I stays awake for most of the time due to multiple outgoing packets and incoming TCP acknowledgments.

Both the throughput and latency for Flow 3 are affected when we use the backbone. The throughput reduces by around 0.80 Mbps because the flow takes one extra hop and experiences additional contention for the shared wireless medium. The latency also increases by ∼60 ms because G is asleep; when J receives packets destined to G, J buffers the packet until G wakes up for the next beacon (sent every 100 ms), learns about and requests the buffered packet. This additional buffering delay, on average, is about half the beacon period. Although we do not report details due to space constraint, we also performed experiments with concurrent flows that share common links. Our results show that the throughput using the backbone was comparable to the network without a backbone.

Finally, we report the control overhead and convergence time of our protocol. For the first five minutes of the protocol, there were 61 control messages per node, with the message size being 56 bytes on average. The interval between periodic heartbeat messages is around five seconds, and the vast majority (about 59 out of 61) of control messages are heartbeat messages. In this 12-node network, it took around 7 seconds to complete the backbone formation, which agrees with the results in Table 1.

Energy Consumption. Lastly, we investigate energy consumption of nodes in different operation modes. For each mode, we measured the battery consumed during a 5-minute interval using ACPI. Since energy consumption can potentially differ depending on the remaining battery capacity, we recharged the battery to its full capacity after each run. We also disabled components such as the LCD screen to min-

imize external effects. As a base case, a laptop without a wireless card on average consumes 595 mWh for five minutes. With a wireless card, a client in sleep mode consumes additional 50 mWh (or 645 mWh total), while leaders and bridges (who are awake) consume additional 105 mWh (or 700 mWh total). For lower-power devices, the effect on node lifetime from such energy savings will be significant.

7. HANDLING MISBEHAVIOR

We discuss various ways in which selfish nodes can deviate from the backbone protocol of Section 4 in an attempt to avoid entering the backbone. We show that such misbehavior that could affect our protocol can be detected and handled via existing punishment mechanisms [3, 19, 21, 26, 27].

Misbehavior. There are several actions nodes could take that would seemingly decrease the amount of time they have to spend in the backbone. First, once a leader or bridge node joins the backbone, it could attempt to simply not fulfill its role, either by not forwarding packets for others or by not attempting to connect to nearby leaders. Also, a node u could send incorrect information to its neighbors to get them to volunteer sooner than they would otherwise or to keep them from asking u to be a bridge node. For instance, u could pretend that all of its neighbors share one leader, making it less likely that u's leader would choose u as a candidate bridge node.

Fortunately, such deviations can be detected. When u sends incorrect information to its leader, u's other neighbors overhear this broadcast and can easily verify if it is correct. They can similarly detect if u never forwards their information to its leader. Watchdog [21] can be used as a general-purpose detector of misbehavior.

Incentive (or Disincentive) Mechanisms. Participants need a way to either punish their neighbors for misbehaving or to provide enough positive reinforcement to keep them from misbehaving in the first place. There are various philosophies of whether to use incentives or disincentives; we do not tread this ground here, but instead note that either can be used in conjunction with our protocol. The punishment (or reward) mechanism is orthogonal to our backbone construction protocol, and need only be used when nodes blatantly attempt to free-ride. In fact, it can even be a completely out-of-band mechanism, such as cash.

We envision our backbone protocol being deployed along with one of the several incentive-compatible forwarding protocols, such as Sprite [26, 27], AdHoc-VCG [3], pathrater [21], or channel jamming [19]. When a node detects misbehavior at the backbone, it can simply push this information up to the higher-layer incentive mechanism, which in turn can dole out punishment. For example, if a node is running pathrater, it could reduce its neighbor's throughput when it detects that it is sending incorrect neighbor information.

8. RELATED WORK

The work we have presented in this paper merges two previously disjoint areas: (1) backbone construction in multihop wireless networks, and (2) protocol design in selfish environments.

The problem of finding a connected backbone has been well studied. Finding a small sized backbone has been the most popular goal [2, 11, 16, 17]. Basagni et al. [5] present simulation results for some of existing backbone schemes and compare the performance in terms of completion time, backbone size, and message overhead. Some other schemes consider node cost such as remaining battery and try to include low-cost nodes in the backbone [10, 24]. In all these approaches, nodes are assumed to be cooperative in the backbone construction, whereas our work constructs backbones in selfish environments.

There exist schemes that consider selfish wireless nodes. Ad hoc-VCG [3] provides a strategy-proof mechanisms for finding a minimum-energy path by carefully determining the reward amount (e.g., using money) for forwarding nodes. Zhong et al. propose a credit-based system in Sprite [26]. They assume the existence of a centralized Credit Clearance Service (CCS). Each node receives a receipt for each packet forwarded, and submits these receipts to the CCS for compensation. Buttyán and Hubaux [8] use a similar approach using virtual currencies, but rely on tamper resistant hardware to store information about the remaining currency. Zhong et al. [27] design protocols that stimulate cooperation for routing and forwarding using cryptographic techniques. Note that all of these approaches require a public key infrastructure for correctness. In contrast, our proposed scheme uses internal incentives only and does not require external money or security infrastructure. As discussed in Section 7, Catch [20] is similar to our proposed scheme in two aspects: (1) it uses internal (dis)incentive of isolation and (2) requires a detection mechanism such as Watchdog [21], which utilizes the broadcasting property of wireless medium for misbehavior detection.

A few recent papers consider the scenario where selfish nodes do not follow the IEEE 802.11 MAC protocol, for example, by using a small contention window. Cagali et al. [9] apply the bargaining game theory to derive an optimal contention window size that each of multiple cheater should use depending on the total number of cheaters. Kyasanur and Vaidya [18] present modifications to the IEEE 802.11 protocol to facilitate the detection of such selfish nodes using RTS and CTS frames. Raya et al. [23] classify different MAC level misbehavior techniques and present a monitoring system that runs on access points to detect and prevent selfish nodes from achieving higher performance. All these protocols consider the issue of a node deviating from the MAC protocol to achieve gain (*e.g.*, higher throughput), which are complementary to our work of building backbones above the MAC layer.

9. CONCLUSIONS AND FUTURE WORK

We have examined how to enforce cooperation using internal incentive mechanisms in a multihop wireless network consisting solely of greedy nodes. We generalize the well-known Volunteer's Timing Dilemma game, based on which we develop an incentive-compatible scheme that constructs efficient routing backbones. Our simulation and implementation results demonstrate that the resulting backbone forms quickly and provides paths and battery savings comparable to protocols designed for fully altruistic nodes.

This paper describes the first backbone construction protocol in selfish wireless networks and naturally leads to a number of open questions: how to deal with node mobility in selfish settings, how to handle node collusion, and how to

form efficient and resilient backbones against potential punishment in the system. Another interesting area of future work will be to design internal incentive based mechanisms that enforces correct end-to-end forwarding over backbones.

Acknowledgments

We thank Aravind Srinivasan and the anonymous reviewers for their helpful comments. This work was partially supported by NSF Awards CNS-626636, CAREER ANI-0092806, and a fellowship from the Sloan Foundation. Dave Levin was also supported in part by a UMD CSD Dean's fellowship.

10. REFERENCES

[1] A. Adya, P. Bahl, J. Padhye, A. Wolman, and L. Zhou. A multi-radio unification protocol for IEEE 802.11 wireless networks. Technical report, Microsoft, MSR-TR-2003-41, June 2003.

[2] K. Alzoubi, X.-Y. Li, Y. Wang, P.-J. Wan, and O. Frieder. Geometric spanners for wireless ad hoc networks. *ACM Trans. on Parallel and Distributed Systems*, 14(5), 2003.

[3] L. Anderegg and S. Eidenbenz. Ad Hoc-VCG: A Truthful and Cost-Efficient Routing Protocol for Mobile Ad Hoc Networks with Selfish Agents. In *Proc. of MobiCom*, 2003.

[4] H. Balakrishnan, C. L. Barrett, V. S. A. Kumar, M. V. Marathe, and S. Thite. The Distance-2 Matching Problem and its Relationship to the MAC-Layer Capacity of Ad Hoc Wireless Networks. *IEEE JSAC*, 22(6):1069–1079, Aug. 2004.

[5] S. Basagni, M. Mastrogiovanni, and A. P. C. Petrioli. Localized protocols for ad hoc clustering and backbone formation: A performance comparison. *IEEE Transactions on Parallel and Distributed Systems*, 17(4):292–306, 2006.

[6] C. Bliss and B. Nalebuff. Dragon-Slaying and Ballroom Dancing: The Private Supply of a Public Good. *Journal of Public Economics*, 25, 1984.

[7] T. X. Brown, J. E. James, and A. Sethi. Jamming and sensing of encrypted wireless ad hoc networks. In *Proceedings of ACM MobiHoc*, 2006.

[8] L. Buttyán and J.-P. Hubaux. Enforcing Service Availability in Mobile Ad-Hoc WANs. In *Proceedings of ACM MobiHoc*, pages 87–96. IEEE Press, 2000.

[9] M. Cagalj, S. Ganeriwal, I. Aad, and J. P. Hubaux. On Selfish Behavior in CSMA/CA Networks. In *Infocom*, 2005.

[10] B. Chen, K. Jamieson, H. Balakrishnan, and R. Morris. Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks. *Wireless Networks*, 8(5), 2002.

[11] F. Dai and J. Wu. An extended localized algorithm for connected dominating set formation in ad hoc wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 15(10):908–920, 2004.

[12] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris. A High-Throughput Path Metric for Multi-Hop Wireless Routing. In *Proc. of Mobicom*, pages 134–146. ACM Press, 2003.

[13] A. Diekmann. Volunteer's Dilemma. *Journal of Conflict Resolution*, 29(4), 1985.

[14] J. Douceur. The Sybil Attack. In *Proc. of IPTPS*, 2002.

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

[16] D. Dubhashi, A. Mei, A. Panconesi, J. Radhakrishnan, and A. Srinivasan. Fast Distributed Algorithms for (Weakly) Connected Dominating Sets and Linear-Size Skeletons. In *Proc. of ACM-SIAM SODA*, 2003.

[17] S. Guha and S. Khuller. Approximation Algorithms for Connected Dominating Sets. In *Fourth Annual European Symposium on Algorithms*. Springer-Verlag, 1996.

[18] P. Kyasanur and N. Vaidya. Selfish MAC Layer Misbehavior in Wireless Networks. *IEEE Trans. on Mobile Computing*, 4(5), 2005.

[19] D. Levin. Punishment in selfish wireless networks: A game theoretic analysis. In *NetEcon*, 2006.

[20] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan. Sustaining Cooperation in Multi-hop Wireless Networks. In *Proc. of NSDI*, 2005.

[21] S. Marti, T. J. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proc. of Mobicom*, 2000.

[22] J. Newsome, E. Shi, D. Song, and A. Perrig. The Sybil Attack in Sensor Networks: Analysis & Defenses. In *Proc. of IPSN*, 2004.

[23] M. Raya, J.-P. Hubaux, and I. Aad. DOMINO: a System to Detect Greedy Behavior in IEEE 802.11 Hotspots. In *Proc. of MobiSys*, 2004.

[24] Y. Wang, W. Wang, and X.-Y. Li. Distributed low-cost backbone formation for wireless ad hoc networks. In *Proc. of ACM MobiHoc*, 2005.

[25] J. Weesie. Incomplete Information and Timing in the Volunteer's Dilemma. *Journal of Conflict Resolution*, 38(3), 1994.

[26] S. Zhong, J. Chen, and Y. R. Yang. Sprite: A Simple, Cheat-Proof, Credit-Based System for Mobile Ad-Hoc Networks. In *Proceedings of IEEE Infocom*, April 2003.

[27] S. Zhong, L. E. Li, Y. G. Liu, and Y. R. Yang. On designing incentive-compatible routing and forwarding protocols in wireless ad-hoc networks: an integrated approach using game theoretical and cryptographic techniques. In *Proc. of ACM Mobicom*, 2005.

Appendix: Derivation of Optimal Waiting Time

Here we derive the expression for the optimal waiting time given in Section 3.2.1. Let $T_v(c_v)$ denote the function for the optimal waiting time for node v, where c_v is v's cost to volunteer. Assume, as discussed in Section 3.2.1, that each node knows its two-hop neighborhood, $\mathcal{N}^2$. For the ease of exposition, let $\mathcal{N}^1(v)$ include only the neighbors of u that have not opted out of the GVTD game. Further, define the set $\mathcal{N}_v^1(u) \stackrel{\text{def}}{=} \mathcal{N}^1(u) \setminus \mathcal{N}^1(v)$, that is, the one-hop neighbors of u that are not also one-hop neighbors of v. Letting $n_u = \left|\mathcal{N}_v^1(u)\right|$, we have:

$$\begin{aligned} Q_v(c_v) &\stackrel{\text{def}}{=} \Pr[v \text{ will have to volunteer}] \\ &= \Pr[\forall u \in \mathcal{N}^1(v) : u \text{ will not volunteer before } v] \end{aligned}$$

A precise formulation of this would require knowing the topology of the entire network, but, as discussed in Section 3.2.1, we can only ensure that a node knows its two-hop neighborhood. Hence we approximate the above event to the event that each neighbor u of v either has greater cost to volunteer or has a neighbor other than v that will allow it to opt out:

$$\begin{aligned} Q_v(c_v) &\approx \Pr[\forall u \in \mathcal{N}^1(v) : (c_v < c_u) \vee \\ &\qquad ((c_v \geq c_u) \wedge \exists w \in \mathcal{N}_v^1(u) \text{ s.t. } c_w < c_u)] \\ &\approx \prod_{u \in \mathcal{N}^1(v)} \left[(1 - c_v) + \int_0^{c_v} (1 - (1-y)^{n_u}) \, \mathrm{d}y \right] \\ &= \prod_{u \in \mathcal{N}^1(v)} \left[1 - \frac{1 - (1-c_v)^{n_u+1}}{n_u + 1} \right] \end{aligned} \tag{3}$$

Bliss and Nalebluff [6] show that the partial derivative of the optimal waiting time $T_v(c)$ with respect to node v's cost c is:

$$\frac{\partial T_v(c)}{\partial c} = -\frac{c}{1-c} \cdot \frac{1}{Q_v(c)} \cdot \frac{\partial Q_v(c)}{\partial c} \tag{4}$$

Using the above approximation of $Q_v(c)$, we obtain the following:

$$\frac{\partial T_v(c)}{\partial c} \approx \sum_{u \in \mathcal{N}^1(v)} \frac{(n_u + 1)\, c(1-c)^{n_u - 1}}{n_u + (1-c)^{n_u+1}} \tag{5}$$

The integral of (5) yields our final expression for $T_v(c)$.