# TinyTorrents - Integrating Peer-to-Peer and Wireless Sensor Networks

Ciarán Mc Goldrick, Michael Clear, Ricardo Simon Carbajo, Karsten Fritsche and Meriel Huggard
School of Computer Science and Statistics
Trinity College Dublin
Dublin 2. Ireland
Email: {Ciaran.McGoldrick, clearm, carbajor, fritschk, Meriel.Huggard}@cs.tcd.ie

***Abstract*—TinyTorrents integrates Wireless Sensor Networks with the BitTorrent Peer-to-Peer protocol and is designed to respect the resource constrained environment which characterises many WSN's. This paper describes the architecture and design which underpins the TinyTorrents system and explores many of the technical challenges that arise in the fusion of such distinct data dissemination networks. The key behaviours and features of both Peer-to-Peer and Wireless Sensor Networks are first explored and evaluated. TinyHop, the protocol used for inter-node routing within the WSN, is presented. Experimental resource optimization and system analysis is performed for message overhead, resource coordination, peer selection and tracker functionality. The system advertises available node and network data via both Web 2.0 service interfaces and through BitTorrent tracker networks. The system has been validated and demonstrated using multiple networks and multiple clients accessing data on a global scale.**

## I. Introduction

Wireless Sensor Network (WSN) and communication technologies underpin many of the advances currently being made in the field of data communications. Such infrastructures seek to make data available in a timely, accurate and secure fashion. The use of wireless sensor networks in unreliable environments may preclude the deployment of persistant network gateways. Instead, mobile temporal gateways can be used to retrieve data from different points of the network (e.g. data mules [1]). The monitoring of glaciers [2] is one such application domain. Unless data is gathered to a gateway continuously, data may be lost when ice blocks break away from the main ice shelf. In this scenario, the deployment of expensive, Internet-connected, gateway devices is intractable and a mechanism to store data within the WSN would prove extremely beneficial. To this extent, TinyTorrents [3] has been proposed by the authors for the reliable, redundant and distributed dissemination of WSN data across the WSN and the Internet. TinyTorrents extends existing peer-to-peer (P2P) methodologies, where data is encapsulated into torrents and spread across the network in a redundant fashion. In the TinyTorrents system, data accessibility is federated and mediated according to both device capabilities and the prevailing needs of end-users. Node and network data and capabilities are characterised and advertised, allowing end-user applications to access, consume and aggregate data from multiple, diverse WSN sources. Torrent clients and Web 2.0 user agents are supported actuators in the current implementation. The TinyTorrents infrastructure currently incorporates a high reliability WSN routing layer (TinyHop) which delivers the bi-directional data communication features required to support the TinyTorrents architecture within the WSN. This paper focuses on evaluating and parameterising torrenting concepts, and the BitTorrent protocol in particular, in a WSN domain. System Integration and WSN power conservation issues form a key element of this analysis. Wireless Sensor Network and Peer-to-Peer concepts are briefly introduced in Section II. BitTorrent behaviours and WSN power management and conservation strategies are then specifically detailed. The TinyTorrents system is then described and the role, purpose and operation of TinyHop documented. The behaviour of specific aspects of the torrenting protocol in a WSN environment are then explored and recommendations provided for future research in the field.

## II. Background

### A. Wireless Sensor Networks

A wireless sensor is a small device that is commonly constrained by both limited battery life and short transmission range. Wireless Sensors can be viewed as producing data, consuming data or collaboratively routing data. Sensor Networks are networks of sensor devices that are characterized by three key features: the ability to gather sensory data about their surroundings; data processing (and possibly discrimination) capabilities; and the capacity to communicate with neighbouring devices [4]. Wireless Sensor Networks (WSN) require that such communications take place wirelessly - most commonly using Radio Frequency technologies. WSN devices (often referred to as motes) expend considerable power engaging in such communication and their available power capacity is often the defining characteristic of their lifespan. Thus much research in the area of WSN's focuses on reducing the power expended in communication, thereby increasing the lifespan of a sensor, or the network as a whole. There are a broad range of WSN platforms and technologies available, with those from XBow systems being most commonly referenced [5], [6]. Mica2 and MicaZ wireless sensor hardware from XBow is used for the live deployment testing of TinyTorrents. These sensors are battery powered and are additionally constrained in that they have very limited program memory, slow CPU's and short radio communication ranges. Consequently system,

978-1-4244-3375-9/09/$25.00 ©2009 IEEE

protocol and power optimization methodologies are of great importance in this environment. Applications and protocols must be designed to be as power efficient as possible. Different techniques have been adopted to reduce the power consumption of both the motes and the WSN as a whole. In [7], three main schemes for energy management are identified: i) Battery Management, ii) Transmission Power Management, and iii) System Power Management. Suggested strategies include the use of sleep mode operation whenever possible, avoiding retransmissions, minimizing control packet size and using power efficient error control techniques. Radio communication is the most power hungry operation that most wireless sensors engage in. The radio is active when transmitting, receiving or eavesdropping on the transmissions of other nodes. The radio also has two other states - idle and sleep. Where possible, the radio should be kept in sleep mode to conserve energy. For instance, the Mica2 radio draws considerably (orders of magnitude) less power in sleep mode than when fully active and communicating [6]. In the same vein, onboard processing and storage operations are much more power efficient than communication operations. These characteristic behaviours of WSN's and their nodes will be used in optimizing the performance of the TinyTorrents protocol.

### *B. Peer-to-Peer Protocols and BitTorrent*

Global data dissemination networks and infrastructures can be realised using Peer-to-Peer (P2P) technologies over TCP/IP. P2P architectures are commonly characterised as moving the resource management to the edge of the network, i.e. to the "peers". This is achieved by allowing direct communication among peers, which encourages them to cooperate amongst each other in managing the resource. The managed resource is most commonly bandwidth, data or computing power. Peers in a P2P system operate autonomously, which means that such systems exhibit greater reliability and availability, as they are more resistant to individual node failures. Peer autonomy also means that peers are often equipped with discovery protocols, which allow them to find functioning peers in the network when neighbouring peers have failed. This ability of "self-organisation" greatly adds to the reliability of P2P systems. It also means that the central node is relieved of much of the management burden. Ideally, a P2P network has no central nodes at all, with decentralised functionality being provided through Distributed Hash Tables (DHT) on all participating nodes. For these reasons, P2P systems generally scale better than corresponding client-server systems. This is most clearly seen in file-sharing applications in which the cost of publishing and transferring data is spread across all the nodes in the P2P network. BitTorrent is a content distribution network based on the peer-to-peer model. BitTorrent is intended for efficient data dissemination on the Internet and was originally designed by Bram Cohen [8]. Its key components are the peers (who cooperate in replicating content across the network) and a tracker (which provides a means for peers to discover each other). BitTorrent allows peers to cooperate directly in transferring data, and ensures fairness among peers using a tit-for-tat algorithm. Content is broken into pieces, each of which are treated independently. This allows different portions of the same file to be downloaded at the same time, and maximises bandwidth utilisation. The goal of BitTorrent is to replicate the content as quickly as possible among the peers, thereby reducing the burden on the original publisher of the content.

The main elements in the BitTorrent architecture are the torrent file, the tracker, the set of peers, and the seed peer. BitTorrent distinguishes peers into two groups: seeders and leechers. Peers which have a complete copy of the file being distributed are called seeders. Leechers are peers which do not yet have a complete copy of the content. The tracker is a centralised component which is responsible for keeping track of all peers currently involved in distribution of a file. It is the means by which peers discover one another, using the tracker protocol which may be layered on top of HTTP. The tracker is not necessarily directly involved in distributing the content. BitTorrent does not include a file search mechanism as other P2P content distribution networks do. Instead, it relies on users acquiring a "torrent" file, which contains metadata about an item of content published through BitTorrent. The torrent file is acquired externally to the BitTorrent system. BitTorrent can be broken down into a number of key elements and behaviours:

*1) Publishing Data:* In order to publish content in BitTorrent a ".torrent" file is constructed. This ".torrent" file is made available outside of the BitTorrent network, typically by hosting it on a standard web server. At least one peer that has the full fileset must be available. This peer is the seed for the torrent.

*2) Tracker Protocol:* The tracker protocol is used between peers and trackers, and signals the intent of a peer to join a torrent. The tracker keeps track of the peer membership in the torrent. Peers attempt to join a torrent using the information about a torrent contained in the ".torrent" file. In particular, the address of the tracker responsible for the torrent is provided. Once a peer has contacted a tracker, the tracker randomly chooses a set of peers from the torrent and communicates this back to the joining peer. Once a peer has joined a torrent, it will periodically inform the tracker of its status and progress. This ensures that the tracker's global view of the torrent is kept up-to-date.

*3) Peer Interaction:* Once a peer has joined the torrent and obtained a list of peers, it proceeds to establish TCP connections with its peer set, over which all peer–to–peer interaction will take place. All peer-to-peer connections are bidirectional. The peer interaction begins with a handshake process. The handshake messages contain a fixed header, followed by the SHA1 hash of a portion of the ".torrent" file, which uniquely identifies the content. If both peers do not agree on this hash, they sever the connection. Once the handshake is complete, the peers begin exchanging data with each other. They do this by interchanging pieces of the file with each other. Whenever a peer completes a piece, it informs all the peers in its peer set. This is important, as it allows peers to keep track of the status of other peers and thus make better decisions regarding piece selection. Two important aspects of

the peer protocol are now highlighted, piece selection and fairness.

*4) Piece Selection:* The piece selection strategy is the mechanism by which a peer decides in what order it should request pieces of the content. In order to select the piece to be downloaded next, peers in BitTorrent use a rarest first strategy. This means that peers choose to download the pieces which the fewest of their own peers already possess. Peers must inform each other when they obtain a new piece. The rarest first approach makes sure that peers have pieces which their peers are likely to want as well as ensuring that the pieces are rapidly duplicated across the network. This means that redundant copies of all pieces are made as quickly as possible, thus reducing the likelihood of content loss due to node failure. The rarest first approach also means that the burden on the initial seed is reduced much more quickly, thus reducing the flash crowd problem [9].

*5) Fairness: Choking and Unchoking:* In BitTorrent, peers essentially barter with each other for pieces of the content. Each peer-to-peer connection in BitTorrent is associated with two states at each end: choked or unchoked, and interested or not interested. The interested status indicates if the peer at the other end of the connection has a piece that this peer still needs. Peers decide to whom they should upload using a tit-for-tat algorithm. They do this using "choking", which is a temporary refusal to upload to a peer. Once a connection is unchoked, then uploading can take place again. In this way peers reciprocate by uploading to peers which upload to them, thus encouraging fairness. Data transfer between two peers will only occur if the sending end of the connection is not choked, and if the receiving end is interested. Maintaining the choking state is key to BitTorrent achieving its goal of fairness as it allows peers to interact with each other and promotes fair participation in the network [10].

### C. Advantages of the BitTorrent design within WSN

Like other P2P content distribution protocols, BitTorrent helps to balance the communication load and avoid network partition. These features correspond exactly to two of the key WSN design goals. BitTorrent also employs the rarest piece algorithm which introduces a higher degree of efficiency in data replication. The BitTorrent approach of segmenting data files into pieces, adds an extra level of inner security to the system. A node which does not contain all pieces of a data file provides only meaningless data when tampered with. Replication policies can be set up so as to avoid peers storing entire data files, thus achieving this added level of security. One additional advantage in using the BitTorrent design is that if every segmented data piece can be transported within a single packet, the probability of successful data delivery in the network increases and, hence, the reliability.

## III. TinyTorrents

The TinyTorrents architecture proposes a new approach to addressing data availability concerns in a sensornet environment. In TinyTorrents, each sensor is an element of the BitTorrent global peer-to-peer (P2P) network[1]. Data is only published to the network when it is requested by a peer, or when the sensor deems it necessary. In the current implementation, trackers and requesting clients maintain state vectors pertaining to the propagation of a published segment. The architecture provides for prioritisation of "rarer" pieces e.g. if imminent sensor failure is likely, or an event of significance has occurred, transfer of the sensor's remaining data can be prioritised. The TinyTorrents infrastructure incorporates a high reliability WSN routing layer (TinyHop) which delivers the bi-directional data communication features required to support the TinyTorrents architecture. TinyHop presumes that any mote can act as a gateway (sink). Mobile or static elements (like data mules [1]) which implement the TinyTorrents system can integrate with the network from any location, thereby helping to balance the traffic load and avoiding energy-based network partition. TinyTorrents operates in tandem with the routing layer, realising a peer-to-peer communication capability between any two nodes in an ad-hoc mesh network. TinyTorrents assumes a highly dynamic environment incorporating node mobility. Thus the routing protocol used must work in scenarios where nodes can be mobile and routes may have to dynamically (re)discovered.

### A. Application Framework

The system presented in [3] interfaces software on a gateway host with a base station mote. The software consists of a plug-in for the popular BitTorrent client Vuze [11], which effectively bridges the TinyTorrents protocol running on the sensor network, with BitTorrent using TCP/IP. An overview of a typical deployment is depicted in Figure 1.

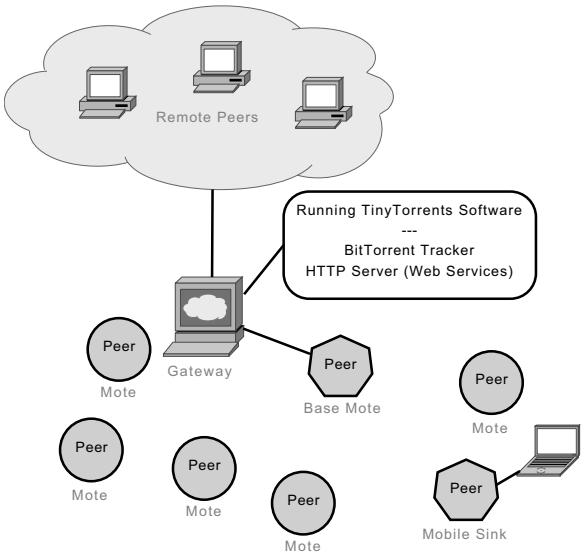


Fig. 1. TinyTorrents System - Globally accessible WSN data.

A gateway connects to the sensor network via a base station. The software running on the node implements the TinyTorrents protocol and may effectively act as a peer in a file transfer.

A file available for transfer in the WSN is described by a concise torrent descriptor containing the number of pieces, the length of the file, the checksums for each piece and the unique key associated with the torrent. The unique key is composed of application-specific metadata. For example, one byte of the key may indicate the type of data (temperature, moisture etc...) and another may indicate the location.

The software framework provides for the attachment of application agents, which provide access to the torrents offered by the sensor network. An Application Programming Interface (API) was designed to expedite the development of such agents. An agent controls which torrents are downloaded from the WSN. For example, an agent designed to collect temperature readings may attempt to transfer all temperature-related torrents published on the network. Alternatively, it may advertise the availability of such torrents (without actually fetching the data) and fetch the data on demand. An agent may additionally aggregate data acquired from the WSN, make appropriate filtrations and make available the fused resultset.

Integration with Vuze offers a multitude of potential functionalities. Vuze is a feature-rich BitTorrent client that incorporates comprehensive tracker functionality. Thus, TinyTorrents-compatible torrents may readily be converted to standard BitTorrent-compatible torrents, and the latter published to the tracker. Therefore, remote users can freely access the data. Such functionality was abstracted from Vuze and incorporated into the API.

Agents may expose available data to internet clients by a variety of techniques. BitTorrent does not provide any integrated querying mechanism so the authors implemented SOAP-based Web 2.0 Services as an effective solution. Facilities for rapidly deploying complex Web Services over HTTP/HTTPS are incorporated into the API. Queries are ordinarily based on the metadata encoded in the torrent's key. Thus an internet user may search for all torrents which include temperature readings at a particular location in a specified sensor environment. An agent running on the gateway may also publish its own torrents on the sensor network. For instance, a critical update may be deployed to the motes in this fashion. However this additionally requires service discovery/querying functionality in the WSN. A multitude of approaches amenable to TinyTorrents that realizes such functionality, are presently being investigated by the authors

### *B. TinyHop*

The TinyTorrents infrastructure relies on the routing layer to perform peer-to-peer wireless communication in unreliable network scenarios. A routing protocol called TinyHop [12] has been designed to operate in an error-prone, lossy radio communications environment and is intended to provide a reliable means of end-to-end communication in as power-efficient a fashion as practicable. TinyHop seeks to minimize the number of messages necessary to perform routing, thus extending the life of both individual nodes and the network as a whole. Careful consideration was given to memory, computational power, and energy constraints in the design. It uses an on-demand paradigm, thereby avoiding the expensive (in power terms) periodic beacon messages of proactive routing protocols [13]. TinyHop is reliable in the sense that it provides bidirectional communication between any two nodes in the network and ensures connectivity through local repair mechanisms. It creates routing paths that work in both directions. It allows any mote to act as an on-demand base station or sink (a node that acts as a gateway and routes data in and out of the network). Base stations in the TinyTorrents system act as gateways to other networks or the Internet. The protocol has been implemented in version 2.0.2 of TinyOS [14] and evaluated both in TOSSIM [15], [16] and in a real testbed of MicaZ [5] motes, which is the platform used for the TinyTorrents demonstrator.

TinyHop uses flooding mechanisms which reduce the flooding overhead associated with typical multipath route discovery, whilst avoiding cycles. In order to achieve bidirectionality and promote fault tolerance, a local flooding mechanism is implemented that seeks alternative bidirectional routes within the neighbourhood of an uncontactable node when route symmetry is lost. A node is considered uncontactable when a timeout semaphore expires. The protocol has been designed to easily incorporate different decision metrics. Such metrics can inform the next hop and local route rediscovery processes. Currently the protocol operates using a fastest route response metric. Packet and route cycles are avoided by design. The protocol also prevents multiple uncontrolled local discoveries when acknowledgment packets are not received and employs retrys and snooping techniques to help prevent the formation of unusable paths.

TinyHop was validated using scalable topologies up to 128 nodes. For each topology, scenarios featuring different degrees of link connectivity were created to simulate local mobility and to force the protocol to employ its local repair mechanism. As the topologies increased in dimension, paths were created from 4 to 23 hops in length. Data such as round trip time (RTT); overhead in sending and receiving; packet loss; and number of data packets sent, received and acknowledged were captured for evaluation of the performance of the protocol. These showed that data packets were always delivered where a bidirectional path existed, although the retransmit rate was seen to increase with hop count. Routes with long hop counts, and elevated intermediate node mobility, required a greater number of local rediscovery processes during end-to-end data transmissions. More impressive results were obtained during validation in a live 32 node WSN deployment, where the actual packet retransmission rate was considerably lower than that indicated by simulation.

As previously noted TinyHop is designed to function in a power efficient fashion on highly constrained sensor devices. It requires one routing entry for every path created in each node. This entry contains minimum information about the route and the next hop. Inactive, or otherwise expired, entries are reused on demand. These design considerations mean that both TinyTorrents and TinyHop work in a cross-layer fashion and require less than 4KB of RAM on a participating node.

### C. TinyTorrents Protocol

TinyTorrents is a contraction of the BitTorrent protocol tailored for sensor devices. Initial design concepts are established in [3]. TinyTorrents preserves the core behaviours outlined in the BitTorrent specification. A unit of data to be shared (file) is subdivided into regular sized *pieces* (save for the last piece). The unit of data is described by concise metainfo, referred to hereafter as a *torrent file*. This *torrent file* serves the same purpose as a ".torrent" file in BitTorrent. It includes information such as the length of the data, the number of pieces, the checksums for all pieces and a unique key. The key is composed of application-specific metadata (as opposed to the SHA1 hash of the shared data used in BitTorrent). Additionally, the key may incorporate the mote's address and a sequence number, to ascertain uniqueness. Finally, the address of the tracker mote is incorporated in the torrent file, i.e. there is no statically selected tracker in the WSN, thus multiple trackers may be deployed to prevent over reliance on a single power-restricted mote.

A "torrent" comprises a set of peers, collectively referred to as a swarm. A peer may be one of two types, seeder or leecher. As mentioned above, a seeder possesses an entire copy of the file and contributes resources to assist other peers; while a leecher has an incomplete subset of available pieces and seeks to procure the remainder, ideally balancing consumption of resources by sharing its pieces with other peers. In order for a leecher to obtain pieces, it requires a peer list (a subset of the swarm). The original BitTorrent specification identified the tracker as the primary regulator of a torrent and as the authoritative maintainer of the swarm, enabling clients to request a list of peers from a centralized location [8]. Thus, the tracker is the only reliable entity that has impartial knowledge of all peers in a given swarm.

In BitTorrent, a tit-for-tat algorithm is employed to ensure fairness. In essence, this is achieved by 'choking' a peer i.e. refusing to provide it pieces until it has uploaded an adequate amount of data. Peers are notified when the choking status for another peer is updated. Such notifications result in heavy traffic in a WSN environment and are thus omitted in the current TinyTorrents system. Alternative strategies are used to preserve fairness within the TinyTorrents environment.

In order for peers to identify nodes in possession of a given piece, BitTorrent requires that each client notify its leeching peers that it now possesses the piece by sending 'HAVE' messages. Such messages generate high volumes of traffic and low-bandwidth peers may be overwhelmed by nominal 'HAVE storms'. Strategies for reducing the frequency of such messages have been explored [17] in the literature. TinyTorrents does not directly use 'HAVE' messages - rather bit vectors encoding the pieces possessed by a peer are employed. For larger files bloom filters have been identified as most appropriate [18]. A leecher exchanges its bit vector with another peer in a handshake phase, which precedes any piece requests made by the former to the latter.

A tracker mote for a torrent has a relatively straightforward behaviour set. It records a mote's address and position in the swarm, and responds to requests for peers with a random subset of the swarm. The entire protocol operates in three phases as follows:

1) Acquisition of a peer list. In the protocols simplest form, a request is made to the tracker and the peer list is established from the tracker's response. In addition, the mote tracks its position in the swarm from the tracker response.
2) Handshakes are performed with all peers wherein piece bit vectors are exchanged.
3) A mote periodically selects the rarest missing piece that is available from its list of peers. If no piece is obtainable, phase 1 is re-entered. Otherwise, a peer, which has the piece, is chosen and a request is sent.

## IV. Protocol Analysis

A thorough investigation of TinyTorrents was undertaken to determine its viability under realistic conditions, particularly its capability to dissipate duties across the network, and avoid excessive reliance on individual motes. In this section, the evaluation environment is briefly outlined, followed by a comprehensive analysis of peer selection with the goal of balancing contributions amongst motes, and minimizing transmissions. The strategies evaluated include those that utilise contribution and proximity as peer selection criteria.

### A. Evaluation Scenario

To test the application-level efficiency of the protocol, an ad-hoc simulator, derived from the TinyOS simulator (TOSSIM), was employed. Various simulations were conducted on randomly generated graphs of 32 nodes consisting of an arbitrarily nominated tracker and 16 designated peers (an initial seeder and 15 leechers). For each simulation, the location of the peers and the tracker were randomized. Ideal routing tables and zero packet loss configurations were provided. As routing and network conditions are thereby obviated, a thorough comparison of the application-level protocol was achieved.

### B. Investigation of Balanced Contribution

Once a mote has determined the rarest piece to acquire, a peer must be selected from which to obtain this piece. In [3], a peer was chosen at random, and with equal probability, from the set of available candidates. This strategy was evaluated over 1000 simulation runs and, as expected, highlights the excessive burden placed on the initial seeder. This can be seen in Figure 2. The x-axis denotes the position at which a peer enters the swarm. Thus the primary seeder is always at position 0. The initial seeder must support full transfer to the first leecher, and potentially contribute $1/n^{th}$ of the support for the $n^{th}$ leecher. As energy conservation is paramount in a WSN environment, compensating for the contribution of seeders prolongs network life. It is therefore desirable to reduce the load on the seeder and dissipate the burden of data transfers and network overhead evenly. Important factors for peer selection include the peers relative position in the swarm,

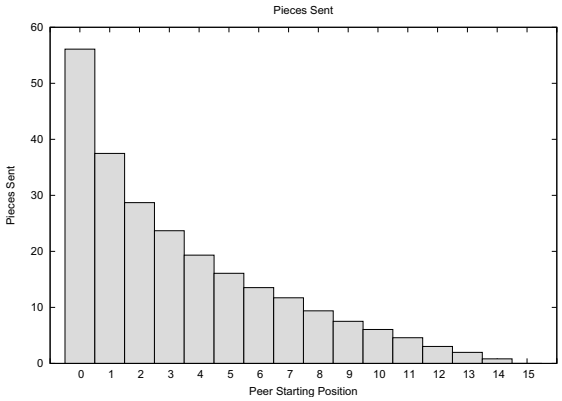


Fig. 2. Peer selection with equal probabilities. The initial seeder at position 0 bears most of the load.

its number of available pieces and the number of concurrent leechers.

One strategy for seeder compensation involves aligning the selection probabilities of each peer with the inverse of its piece count. However, this method is overly simplistic and is influenced by the number of simultaneous leechers. For example, if a leecher enters a swarm of two seeders, the piece count of the latter nodes will be equal and both seeders are chosen with equal probability. The older (possibly original) seeder is not differentiated and therefore does not benefit from a proportionate reduction in load. Hence, piece count alone is an insufficient criterion. Much improved results emerge from basing the probabilities on peer position. Figure 3 demonstrates the outcome of simulations to evaluate the effect of peer position sensitive strategies. The reduced load

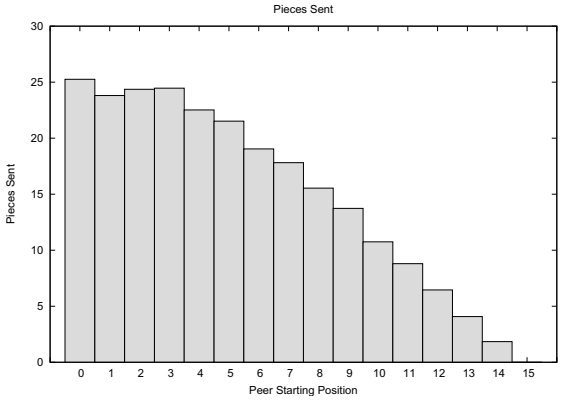


Fig. 3. Peer selection with probabilities proportional to starting position

on the seeder is evident, transmitting an average of 25 pieces as opposed to 55, see Figure 2. The 95% confidence interval for the mean is $21.98 - 28.12$, while the standard deviation is 1.9. In Figure 3, there is a noticeable peak at position 3 arising from the maximum permissible size of a peer list being set to 4 in the simulations. In general for a maximum peer list size $s$ and position $p$, increasing the sensitivity of the probabilities to $p$ results in a peak in the region where $p = s$. In the figure, $s = 4$, and the weight function which ranks each peer is the power function $p^2$. The weight function performed well when defined as a power function $w$ of $s$ and $p$, as follows:

$$w(p, s) = p^{\log(s)}$$

A tracker notifies peers of their position in the swarm through ordinary peer list responses, and such information is exchanged amongst peers in handshakes. Motes are therefore afforded the opportunity to dynamically adjust seeding duties in accordance with declining battery life or excessive piece requests. Under normal circumstances a mote is expected to indicate its actual status to ensure overall network fairness. The Figure 3 findings resulted in some modifications to the protocol. Firstly, the tracker was adapted to record the position of each joining peer in the swarm, and to include this information in peer list responses. Secondly, the handshake phase was modified to incorporate the positions of both peers. Finally, the weight function derived from the analysis was factored into the peer selection strategy employed by the motes.

Reducing the burden on the primary seeder for wired networks has been explored for BitTorrent. A super-seeding technique was evaluated in [19], this ensures that the initial seeder only sends a single copy of each piece. Likewise, the approach adopted here significantly reduces the probability of the initial seeder sending duplicate pieces. To completely eliminate such a likelihood would be inappropriate as the availability of all, or any, of the secondary seeders can not be assured in a mobile communications environment. Moreover, the above approach offers an acceptable solution to the flash crowd problem. This problem arises when a resource achieves sudden unexpected popularity, resulting in excessive demand on the provider, and possibly incapacitating the node. Its effects have been studied in the context of BitTorrent in [20]. Our results demonstrate rapid dissipation of duties amongst peers in TinyTorrents thereby reducing the likelihood of a mote being overwhelmed by an unprecedented burst of interest.

In a WSN, location-based decisions are fundamental to energy conservation. Choosing peers based on hop count, or other proximity metrics, can reduce the number, and power, of transmissions. As evident in Figure 4, the number of packets sent grows linearly with the size of the network. However, there is a trade-off between overall reduction in transmissions and the load borne by each mote. Factoring hop count into the above weight function determination further reduces the total transmissions whilst preserving load balancing functionality. However, the reduction in transmissions was only 3% on average. When the additional complexity arising, and slight reduction in packets sent, are considered together, location-based selection strategies are not included in the implementation of the TinyTorrents protocol described herein.

## V. System Integration

Once the key system messaging principles and data structures for the TinyTorrents system were established, it was integrated with TinyHop using TinyOS version 2.0.2 [14]. Simulations were executed in TOSSIM, the TinyOS discrete

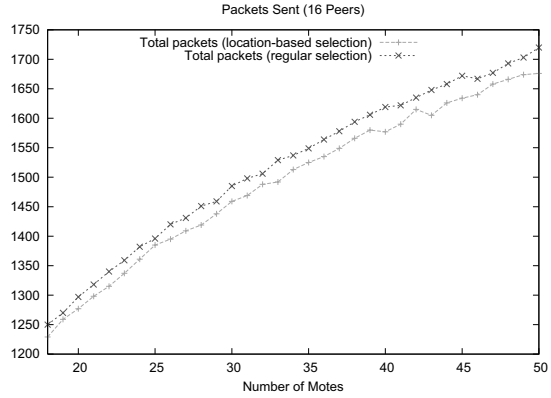


Fig. 4. Variation of transmissions with the size of the network for both regular selection and location-based selection (probabilistically favouring nearby peers)

event simulator using the Crossbow MicaZ platform [21]. A key initial goal was to determine the data efficiency of the system i.e. deduce the implicit TinyTorrents overhead.

### A. Management Overhead

Experiments were carried out to determine the overhead of the protocol i.e. the aggregate amount of control bytes transmitted. The maximum theoretical data vs. control efficiency of a request/receive protocol using single-hop TinyOS active messages is 64.4% [3]. Extra overhead is incurred for the routing layer in multihop paradigms. Irrespective of such invariable additions, the amount of application payload arising from various stages of the protocol was investigated. A topology of 12 motes was configured as shown in Figure 5. One was designated as the tracker, another as a primary seeder and a further 5 as leechers. TinyHop was employed at the routing layer. Three sets of tests were performed requiring torrenting of files of 4, 8 and 16 pieces respectively. The size of each piece was 16 bytes. The entry of leechers into the swarm was randomized for each simulation, as this factor governs the concurrent number of leechers and therefore affects medium contention and interference characteristics. Simulations were repeated 20 times and the results averaged. The results of the simulations for transfers with different numbers of pieces are presented in Table I. The third configuration, based on a

TABLE I
PERCENTAGE OF CONTROL BYTES VS. DATA IN THE APPLICATION PAYLOADS OF TORRENT TRANSFERS INVOLVING 4, 8 AND 16 PIECES

| | Percentage Bytes | | |
|---|---|---|---|
| | 4 pieces | 8 pieces | 16 pieces |
| Requests to tracker | 3.14 | 1.82 | 0.99 |
| Responses from tracker | 7.29 | 4.25 | 2.38 |
| Handshakes | 16.43 | 9.47 | 5.23 |
| Piece Requests | 12.74 | 15.26 | 17.13 |
| Pieces | 60.41 | 69.18 | 74.27 |

torrent of 16 pieces, resulted in a high volume of simultaneous transmissions, thereby providing a challenge for TinyHop's failure recovery modalities. End–to–end delivery was the key

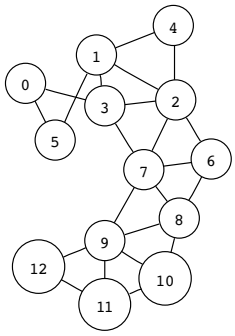


Fig. 5. 12-node Topology. The edges indicate bidirectional reachability

metric of interest from an application perspective. The data in Table I shows the gain in efficiency that results from files with more pieces. This characteristic behaviour is shared with BitTorrent. The percentage of bytes transmitted to and from the tracker in an entire torrent transfer is just 3.37% for 16 pieces and decreases in inverse proportion to the number of pieces. Therefore, the load on a tracker mote is relatively insignificant. The findings (in Table I) reinforced several design decisions pertaining to the protocol. In particular, the overheads associated with the handshake phase decrease significantly in accordance with the number of pieces. The routing layer and Active Message header overhead imposed by TinyOS accounted for 54% of all bytes transmitted, the remaining 46% being apportioned as in Table I. Much of that arises from the initial flooding to establish the primary network topology. Clearly, overheads on this scale are difficult to justify or sustain in a WSN environment. Consequently, further optimization approaches were examined.

TABLE II
AVERAGE PIECES SENT PER SIMULATION OF A COMPLETE TORRENT TRANSFER INVOLVING 5 LEECHERS FETCHING 16 PIECES. IN THE SECOND SIMULATION BATCH, PIECES WERE ADDITIONALLY CACHED EN-ROUTE

| | Mean Pieces Sent | 95% Confidence Interval for the Mean | Standard Deviation |
|---|---|---|---|
| Simulation Batch 1 (No interception) | 93.68 | 89.82 - 97.54 | 4.29 |
| Simulation Batch 2 (Interception) | 76.71 | 68.66 - 84.76 | 6.06 |

### B. Interception and Aggregation

As all motes in the WSN may serve as routers, traffic that is being forwarded may be cached by intermediary nodes if required. In particular, if a mote is a leecher for some torrent and receives a piece of the file which is in transit to another mote, then storing (or caching) this piece may reduce future interactions with the seeder. To establish the potential benefit of this technique, and assess its likely impact, slight modifications were made at the routing layer to pass en-route packets of a specific type to the application. A 12-mote topology was employed comprising a tracker, primary seeder

and five leechers. Assignments were randomized for each simulation. In order for the 5 leechers to acquire a full file of 16 pieces, the minimum number of pieces that must be sent is $80(16\times5)$. Two batches of 20 simulations were performed. The second batch exploited the interception technique described above. Table II establishes the mean reduction in pieces sent as approximately 18%, a considerable reduction in messaging. The 95% confidence interval for the mean for both batches of simulations is provided. This confidence interval is, naturally, wider for the second batch of simulations since the randomized positioning of leechers per simulation exclusively governs the extent of interception. Nevertheless, the upper limit of this interval is appreciably less than the lower limit of the interval for the first batch of simulations, and consequently affirms that reasonable gains are attainable by employing this approach. Exploiting this technique offers many opportunities to reduce the aggregate network and per node power consumption.

## VI. Future Directions

TinyTorrents offers considerable scope for improvement. A major weakness of the protocol, as described herein, is its reliance on a tracker. Decentralization is a significant goal in the pursuit of fault tolerance and scalability. [22] establishes that Distributed Hash Tables (DHT) are not an efficient option for sensor networks domains. An alternative strategy for peer discovery, known as Peer Exchange (PEX), has been implemented in some BitTorrent clients [23]. PEX entails querying present peers to find others. In this regard, it has some parallels with the presented caching behaviour. It is, however, susceptible to the formation of articulation points, whereby the failure of a peer partitions the swarm into disjoint components. It thus becomes necessary to flood in order to re-discover remaining peers. Our research suggest the deficiencies of PEX are more pronounced in the context of TCP/IP networks than for WSNs, as the existence of a recovery mechanism, albeit a suboptimal one, within TinyTorrents offers a functional solution where the probability of partition is reasonably low. There is also considerable scope for community contribution in the area of efficient, integrated querying. Investigation of more efficient routing protocols than TinyHop, which incorporate inexpensive querying functionality, is ongoing. Efficient compression mechanisms, like Bloom filters, are being employed to replace the pieces bit vector; while the impact of both piece length and the number of segmented pieces per data file on system performance is being studied.

## VII. Conclusion

This paper presents TinyTorrents as a means of interfacing Wireless Sensors and Wireless Sensor Networks with the global Peer-to-Peer network, BitTorrent. The background and rationale underpinning the conception and design of TinyTorrents is established and a strategy for interfacing the platforms presented. TinyHop, the protocol that underpins inter-node communications in TinyTorrents, is introduced and its key features summarised. Key messaging behaviours required by BitTorrent are assessed and corresponding functionality documented in TinyTorrents as appropriate. Specific peer and seeder selection strategies are evaluated in a WSN context prior to a system level evaluation of the overall scheme. The resulting optimisations focus on reducing the number of messages and transmissions required by the system – thereby improving network and node lifetime.

## Acknowledgment

This publication has emanated from research conducted with the financial support of Science Foundation Ireland.

## References

[1] D. Jea, A. A. Somasundara, and M. B. Srivastava, "Multiple controlled mobile elements (data mules) for data collection in sensor networks," *DCOSS*, pp. 244–257, 2005.

[2] M. Heavner, R. Fatland, E. Hood, C. Connor, M. Habermann, and L. Gerner, "Monitoring Lemon Glacier Using a Wireless Sensor Network in Juneau, Alaska: The SEAMONSTER Project," in *Joint Meeting of the Geological Society of America*, October 2008.

[3] K. H. Fritsche, "TinyTorrent: Combining BitTorrent and SensorNets," TCD, Tech. Rep. TCD-CS-2005-74, 2005.

[4] J. M. Hellerstein, W. Hong, and S. R. Madden, "The sensor spectrum: Technology, trends and requirements," *SIGMOD Record*, vol. 32, 2003.

[5] Crossbow Technology Inc., "MicaZ Specs," December 2007.

[6] ——, "Mica2 datasheet," October 2008.

[7] S. Jayashree and C. S. R. Murthy, "A Taxonomy of Energy Management Protocols for Ad Hoc Wireless Networks," *IEEE Communications Magazine*, vol. 45, no. 4, pp. 104–110, April 2007.

[8] B. Cohen, "Incentives Build Rebustness in BitTorrent," in *Workshop on Economics of Peer-to-Peer systems*, May 2003.

[9] L. Guo et al, "Measurements, Analysis and Modeling of BitTorrent-like Systems," *Internet Measurement Conference*, 2005.

[10] R. Thommes and M. Coates, "BitTorrent fairness: analysis and improvements," *Proc. Workshop Internet, Telecom and Signal Proc, Noosa, Australia*, 2004.

[11] Vuze. (2008, October) Bittorrent client. [Online]. Available: http://www.vuze.com

[12] R. Simon Carbajo, M. Huggard, and C. Mc Goldrick, "An End-to-End routing protocol for Peer-to-Peer communication in Wireless Sensor Networks," *ACM MiNEMA Workshop*, April 2008.

[13] V. Raghunathan and C. Srivastava, "Energy-aware wireless microsensor networks," *IEEE Sig. Proc. Magazine*, vol. 19, no. 2, pp. 40–50, 2002.

[14] TinyOS Alliance, "TinyOS version 2.x," October 2008, http://www.tinyos.net/tinyos-2.x/.

[15] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: accurate and scalable simulation of entire tinyOS applications," *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, pp. 126–137, 2003.

[16] H. Lee, A. Cerpa, and P. Levis, "Improving wireless simulation through noise modeling," *Proceedings of the 6th international conference on Information processing in sensor networks*, pp. 21–30, 2007.

[17] Huang Daoying et al, "MultiHave: The Extension of Peer Protocol in BitTorrent System," in *Fifth International Conference on Grid and Cooperative Computing Workshops*, 2006.

[18] A. Broder and M. Mitzenmacher, "Network applications of Bloom Filters: A survey," *Proc. of Allerton Conference*, 2002.

[19] Z. Chen, "Experimental Analysis of Super-Seeding in BitTorrent," in *IEEE International Conference on Communications*, China, 2008.

[20] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, "A measurement study of the bit-torrent peer-to-peer file-sharing system," Tech. Rep., May 2004. [Online]. Available: http://pds.twi.tudelft.nl/ pawel/pub/bittorrent.pdf

[21] E. Perla, A. O'Cathain, R. Simon Carbajo, M. Huggard, and C. Mc Goldrick, "PowerTOSSIM z: Realistic Energy Modelling for Wireless Sensor Network Environments," in *3rd ACM International Workshop PM2HW2N*. Vancouver, Canada: ACM, October 2008.

[22] Araujo et al, "4.CHR: A Distributed Hash Table for Wireless Ad Hoc Networks," in *Proceedings of the Fourth International Workshop on Distributed Event-Based Systems (DEBS) (ICDCSW'05)*, vol. 04, 2005.

[23] Vuze. (2008, October) Peer exchange. [Online]. Available: http://azureus.aelitis.com/wiki/ index.php/Peer Exchange