Drift: A Highly Condensed Emulation Framework for Mobile Nodes in Server Clusters

Xinyu Zhang, Baochun Li*
Department of Electrical and Computer Engineering
University of Toronto
Email: {xzhang, bli}@eecg.toronto.edu

Abstract

Prototyping application-layer algorithms in wireless networks is a lengthy and challenging process, involving either programming within a specific simulation platform, or deploying a real testbed that is neither flexible nor scalable. In this paper, we present Drift, a high-performance wireless emulation testbed that takes advantage of the benefits of both simulation and real implementation, while trying to avoid their drawbacks. As a highly condensed emulation infrastructure, Drift makes it possible to rapidly develop and validate large-scale wireless network application-layer protocols within a cluster computing environment. Unlike existing emulation testbeds, Drift features a fully decentralized architecture, an efficient message processing unit, and more accurate network models for mobile wireless nodes. It balances the fundamental trade-off between scalability and emulation accuracy, focusing on maximizing scalability with minimal loss of accuracy. Through baseline comparison and extensive experiments, we find that Drift is able to accommodate thousands of emulated nodes per server host (in contrast to only tens of nodes typically seen in existing emulation tools), while maintaining comparable accuracy to packet level simulators. Drift will be released as an open source platform.

1. Introduction

It has long been the academic tradition to use simulation, emulation, or real-world experimental testbeds to evaluate application-layer algorithms in wireless networks. Different experimental methodologies differ significantly with respect to their scalability, accuracy (when compared to reality), flexibility, as well as cost. Simulations may deviate from reality, but they are less costly, more flexible, and more scalable than real-world experimental testbeds, which produce more realistic performance results. Emulation testbeds achieve the middle ground, and combine the advantages of both simulations and real-world testbeds by emulating wireless transmissions over wireline networks, using real operating systems and network protocol stacks of the networked hosts. Unfortunately, emulation methodologies also inherit some of the drawbacks.

Contemporary wireless network emulators have limited scalability. They tend to adopt one or both of the following paradigms. First, a centralized structure [16, 21, 23], consisting of a dedicated controller and a certain number of slave hosts. All the network events (pairwise packet transmission, node position update, etc.) have to be managed by the central controller. Obviously, such an arbitrator may become a communication bottleneck, when intensive traffic is routed through it. This is why such emulators fail to work under the load of tens of nodes and several MBytes/second traffic, which is far from enough to satisfy the growing scale and in-network processing requirements in current wireless networks. Yet another paradigm is the single-node emulation [9, 21], where each physical machine emulates one wireless node, which apparently does not scale. Beside lack of scalability, unduly simplified network models and the complexity of configuring the OS kernel — as required by most existing emulators — also upset potential users.

In this paper, we seek to solve the above problems with *Drift*, a high-performance emulation testbed which distinguishes itself with a fully distributed architecture, lightweight network models as well as other design choices that aim at scalable emulation of wireless networks with minimal loss of accuracy. With *Drift*, thousands of nodes can be conveniently deployed to a server cluster, running customized *application-layer* algorithms and protocols. The *Drift* infrastructure is installed as an application program and therefore needs no configuration of the emulation hosts' kernels. As a fundamental design objective, *Drift* is meant to scale well in a clustered environment. It is able to support more than

*This work was supported in part by LG Electronics, and by Bell Canada through its Bell University Laboratories R&D program.

one thousand nodes in a single host, which is already beyond the capacity of centrally controlled emulators. An even more invaluable property is that its scalability increases linearly with the number of hosts, which is achieved by minimizing the communication overhead between hosts, and by localizing all the parameters of an emulated node. With a fully distributed architecture, we remove the potential capacity restrictions imposed by a central arbitrator, and distribute the computation load throughout each host in the cluster. We have also planned to release *Drift* as an open source platform, with the hope that it may become beneficial to other researchers in the community.

Modeling the lower layers has been a formidable barrier for the research community in wireless network emulation. Existing emulators either avoid this obstacle [21, 22] or try to provide lengthy details [23] at the cost of efficiency. *Drift* balances the trade-off between scalability and accuracy using an abstract MAC layer that handles the transmission schedules of neighboring nodes in the unit of groups, as well as an analytical MAC layer model that is tailored for further accuracy requirements. As to the physical layer model, we feed empirical data into an accurate analytical model to capture the path-loss effects. With these measures, *Drift* achieves a level of fidelity comparable to detailed packet level simulations, while leaving sufficient room for improving scalability.

The remainder of this paper is organized as follows. In Section 2, we describe in detail the composition and design highlights of *Drift*, including our model of the wireless network stack, and our implementations that enable the distributed architecture. Section 3 evaluates the performance of *Drift* with respect to scalability and realism through baseline comparisons and extensive experiments. In Section 4 we compare our work with existing literature in wireless network emulation. Finally, Section 5 concludes the paper.

2. Drift: Design and Implementation

As a cornerstone of *Drift*, *scalability* has been our primary design objective from day one. Existing emulation testbeds does not support more than tens of nodes with heavy traffic, which is insufficient to understand the behaviors of application-layer algorithms in large-scale wireless networks [7]. To achieve high scalability, one needs to minimize the overhead introduced by the emulation infrastructure, and to be able to accommodate complex algorithms — such as network coding — with a light-weight testbed. Scalability, however, implies simplicity, which does not bode well for accurately emulating reality. In order to be scalable but with sufficient realism, we have to face the following challenges:

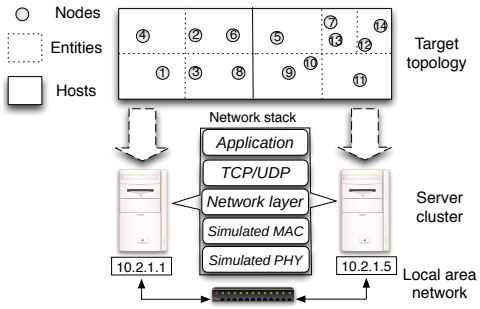

Figure 1. An outline of the Drift framework.

Efficient implementation of a distributed architecture. To represent the shared nature of the wireless environment, each emulated node must efficiently sense the activities of neighboring nodes (possibly in different hosts) without consulting a central authority. To enable distributed emulation, the lower layer models must also be distributed and can be efficiently implemented. In addition, pairwise transmissions and the corresponding bandwidth constraint must be directly realized, without going through a central host.

Modeling MAC layer. Although it generally produces more reliable results, detailed packet-level simulation of MAC protocols involves highly frequent message exchanges which, unlike that of simulators, costs considerable communication overhead. We seek to avoid such message exchanges , while still conforming to the general behaviors of a real MAC.

Modeling physical layer effects. To model the path-loss effects in a realistic manner, the emulator must remove the assumption of perfect transmission within a circular range, without degradation of scalability.

Coping with mobility and other network dynamics. To ensure scalability, each node has to autonomously compute their locations in real time, rather than resorting to a central authority. On the other hand, node mobility invites a significant level of network dynamics, such as frequent connection failures and reset, which have to be handled efficiently by the emulator.

In the following subsections, we present more details on the design of *Drift*, which aims to solve the above problems.

2.1. Drift architecture

Drift features a decentralized architecture, in which multiple physical machines (*i.e.*, emulation hosts) assume equal roles and connect to other hosts through a Gigabit Ethernet switch (Fig. 1). In each emulation host, we create multiple instances of the *Drift* program, each running as a single-process multi-thread *entity*. These entities have the same IP address with the emulation host, but differ in their range of port numbers. Each entity can accommodate several hundred wireless nodes.

A node must have the same IP address as its entity and its port numbers (including UDP control, UDP data, and TCP listening port) are assigned according to the port number range of the entity. In general, the emulation host's kernel restricts the number of open sockets (*e.g.*, 1024 in Linux) in each process. Since each node takes up at least 3 ports, the maximum number of nodes in an entity is quite limited (*e.g.*, at most 341 in Linux). By allowing multiple entities in one host, however, we can fully explore the per-host computation and communication capacity.

Given a target wireless network topology, we map the nodes to the *Drift* framework according to their geographical locations (Fig. 1). Specifically, we split the topology into multiple subareas, each represented by a single emulation host. A host's topology can be further split into multiple subareas, each containing multiple nodes and emulated by a single entity. Note that different host's or entity's area may not be equal. We assign more entities to the area with high node density, if the topology is prescribed and static. We are in the process of designing a dynamic load balancing scheme for time-varying topologies with highly unbalanced node distribution.

The internal architecture of an emulation entity consists of three parts (Fig. 2): the wireless algorithm, the wireless network model, and the message switch. In a general sense, each of them corresponds to the application layer, MAC&PHY layer, and TCP/IP layer in the OSI model. Specifically, the algorithm generates application data, processes incoming packets, and tags them with additional information. Processed packets are then pushed into an outgoing buffer. The message switch is responsible for transmitting and receiving packets in the outgoing buffer through TCP/UDP sockets. Before the actual transmission, each packet is subject to the MAC scheduling and physical layer per-link bandwidth constraint, which are managed by the wireless network models.

In order to improve the efficiency of emulation, we allow nodes in the same entity to share the message processing unit. We observe that memory usage is not a problem in most cases (Sec. 3), thus each node maintains its own memory-demanding data structures, such as the incoming buffer and outgoing buffers. Consequently, an emulated node becomes a set of self-contained variables describing its status and properties, and all nodes in the same entity share the same functions that regulate packet transmission/receptions, manage the incoming and outing buffers and simulate the lower layer models. In the following sections, we provide more details on these functionalities that are critical to the performance of *Drift*.

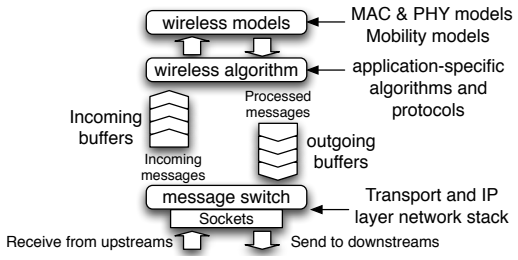

Figure 2. Internal architecture of an entity.

2.2. Wireless network models

In general, wireless network emulators take advantage of the transport and IP protocols in the emulation hosts themselves, but the MAC and PHY models are still simulated. For centralized emulation systems (see, *e.g.*, [16,21–23]), it is possible to simulate the MAC and PHY in the central controller using discrete event simulators like ns-2 and Qualnet [11]. Obviously, such an approach does not apply for distributed emulation. To allow the emulation hosts and entities to locally manage the lower layer behavior, *Drift* adopts two kinds of MAC models: an *abstract MAC* and an *analytical MAC*. The physical layer effects are simulated through an analytical framework based on empirical parameters. We also incorporate application-layer routing protocols and a comprehensive class of mobility models, which are introduced below.

2.2.1 The MAC layer models

Emulating the actual interactions of nodes under a real MAC protocol will undoubtedly compromise the scalability and even the real-time nature of an emulator, because of the highly frequent message exchanges and the different time scales between wireless and wireline transmissions. We avoid the complex message exchanges with two kinds of MAC models: the abstract MAC which jointly manages each group of nodes sharing the same channel, and the analytical MAC which derives the per-link bandwidth (*i.e.*, achievable rate) through mathematical models.

Abstract model. The abstract model features a coarse-grained simulation of the wireless MAC layer. The key observation underlying this scheme is that the exact trace of the MAC layer message exchanges is not important. What really matters is the estimation of achievable throughput between any transmitter-receiver pairs at any given time. To model the bandwidth allocation between competing nodes by the MAC layer, we partition the network into groups centered around each active transmitter. The group size equals to the interference range. A node cannot receive packets if it falls in the interference range of an interfering node and the total

incoming throughput of all group members must be less than the channel capacity. The packet reception of each member is enabled in a randomized round-robin fashion (by the message switch) in order to enforce MAC layer fairness. Such an abstract MAC model corresponds to an ideal scheduling scheme in which interfering nodes can optimally multiplex the channel.

As a general model that captures the wireless interference and bandwidth competition, the abstract MAC is appropriate for the following cases. (1) Prototyping applications which are not sensitive to the lower layer models [10], or algorithms that are based on ideal scheduling models [15]. (2) Highly scalable emulation where the number of nodes and computation efficiency are of a higher-priority concern. A case in point is to understand the benefits of randomized network coding in wireless mesh networks [4].

Analytical model. Occasionally, one may want to explore the influence of MAC layer settings on the target application. We therefore incorporate an analytical MAC model which offers higher accuracy than the abstract model without significant sacrifice of scalability. The analytical MAC is based on existing work on modeling the link throughput in 802.11 multi-hop wireless networks [8]. This mathematical model acts as a plug-in module in *Drift*. Whenever topology changes or a new transmitter-receiver pair starts, the module re-computes the MAC level bandwidth of each active link.

The analytical model eliminates the overhead due to real MAC layer interactions, but it still involves complex computations. The computation typically takes tens of seconds, for a network with several hundred active links. Apparently, the time scale of such computation must be less than the time scale at which the topology changes. Therefore, the analytical model is best applied to scenarios where nodes are static or moving at a low speed, and where the flow structure does not change with high frequency.

2.2.2 Physical layer model

Simply put, the physical layer model in *Drift* predicts link bandwidth based on the distribution of Received Signal Strength (RSS) at a certain distance. Empirical measurements have shown that RSS is a good predictor of link bandwidth, while the distribution of RSS at a certain distance can be approximated by choosing appropriate parameters for the log-normal shadowing model [3]. Based on these observations, we develop a light-weight analytical model that approximates link bandwidth given the link distance. With the shadowing model, we remove the assumption of perfect reception within transmission range, which is commonly assumed by existing emulators (*e.g.*, [16, 21]). Unlike the empirical measurement which is limited to fixed settings (data rate, packet size, *etc.*), we take the following steps to derive the link quality.

The RSS (in dBm) at a certain distance is given by the well-known shadowing model:

$$P(d) = P(d_0) - 10\beta \log\left(\frac{d}{d_0}\right) + X \qquad (1)$$

where d_0 is a reference distance with known RSS; β is the path-loss exponent; X is a Gaussion random variable with zero mean and standard deviation σ. β and σ are obtained from empirical measurements [3].

Denote T as the empirical RSS threshold, then the physical layer reception probability can be derived from (1) as:

$$\Pr(P(d) > T) = \frac{1}{2} erfc\left(\frac{T + 10\beta \log\left(\frac{d}{d_0}\right) - P(d_0)}{\sqrt{2}\sigma}\right)$$

where *erfc* denotes the complementary error function. Since the shadowing time scale is usually much larger than the duration of a packet transmission, it is reasonable to assume the MAC layer ACK and DATA packets have the same reception probability. Then the probability of a transmission failure due to DATA or ACK packet loss is given by:

$$p = 1 - (\Pr(P(d) > T))^2$$

Previous work [8] has modeled 802.11 link layer throughput under collisions but without channel impairment. Here we consider the link throughput with non-ideal physical channels. In particular, under the 802.11 CSMA model, the probability of transmission at a time slot is given by [8]:

$$\tau = \frac{2q(1-p^{m+1})}{q(1-p^{m+1}) + W_0(1-(2p)^{(m+1)}(1-p))}$$

where $q = 1 - 2p$; m is the retransmission limit; W_0 is the minimum back-off window size. Since the PHY model only accounts for the bandwidth of a single link, the channel is either idle or occupied by a transmission attempt. Therefore the link bandwidth equals to the expected amount of traffic that is successfully delivered within in unit time, *i.e.*,

$$T_p = \frac{\tau \cdot (1-p) \cdot \textit{PacketSize}}{\tau \cdot T_s + (1-\tau) \cdot \textit{SlotTime}}$$

where T_s is the duration of a successful transmission and can be approximated with:

$$T_s = \frac{\textit{Data Size} + \textit{ACK Size} + \textit{Header Size}}{\textit{Link Capacity}}$$

The notion *Link Capacity* denotes the maximum throughput of a wireless link, *e.g.*, 2MB/s in 802.11 basic mode. The gap between the link throughput T_p and the link capacity is solely due to the packet losses caused by channel impairment. From the perspective of higher layer protocols, the direct consequence is random packet losses, whose probability can be computed as the ratio of T_p over link capacity.

The above physical layer model accounts for packet losses caused by large-scale path-loss. Such per-link losses are observed to be independent of each other [1]. However, the actual emulated per-link throughput in a multi-hop network may be much lower than the achievable link bandwidth, due to interferences between links, which have been accounted for by the MAC layer models. In addition, our model can incorporate more realistic channel behaviors by replacing the link-distance based reception model with empirical traces.

2.2.3 Routing protocols

It is impossible to implement the large collection of existing application-layer routing protocols. For many applications, routing is not even a necessary component. In view of this, we only support simple geographical routing protocol and link-quality based shortest path routing protocols [5] in *Drift*. Though we are in the process of implementing other routing models, application developers may easily extend the default routing protocols with the interface we provide. Here we emphasize again that *Drift* aims to provide a light-weighted infrastructure for prototyping new applications, rather than to exhaust all protocols in literature.

2.2.4 Mobility models

We build a comprehensive class of mobility models, including trace based and random trip models. In the former model, nodes' location changes may be specified in trace files following prescribed format. The latter include random waypoint, random walk with wrap, random walk with reflection, random walk on general domain, and random walk in a city map. It has been proven that constructing random-trip models in a naive way may result in unrealistic models due to the "initial transition" problem [2]. We therefore construct the whole class of random trip models based upon theoretical work on the perfect mobility simulation [2], which screens off such effects in order to produce unbiased evaluation results. To reduce computation intensity, mobile nodes update locations in a periodical way, and the period can be configured according to the granularity requirements of specific applications.

2.3. The message switch

The message switch provides an interface between the wireless algorithm and the TCP/IP layers in the emulation host's kernel. It consists of two core components called the *network* and *engine*, respectively. The *network* serves as a high-speed packet transceiver which accepts TCP/UDP connections from upstream nodes, buffers incoming packets, and sends the processed packets to downstreams. The *engine* extracts packets from the incoming buffers and switches them to the wireless algorithm for further processing. After possibly complex processes (*e.g.* packet encoding and decoding) by the algorithm, the outcome is switched back into the outgoing buffers for transmission.

A major task of the message switch is to emulate the bandwidth caps imposed by the MAC model and the packet losses imposed by the PHY model. The bandwidth emulation component lies within the *network* module and is built atop the UNIX `select()` function. Within each emulation entity, all incoming and outgoing TCP/UDP connections are monitored by a single `select()` call with a specific timeout value. The actual transmission or reception happens only when the corresponding sockets is ready, and a socket is put into the set of socket descriptors for `select` only when the corresponding MAC layer bandwidth constraints are relaxed. Similar mechanism is applied to emulate the per-link delay constraint. Packet losses due to channel impairment are emulated by randomly dropping packets in the outgoing queue, according to the loss probability observed by higher layers (Sec. 2.2.2).

To facilitate the emulation of data sources in live sessions, the message switch provides a testing data source, which generates data from a regular file, the standard input, or a stream of randomly generated bytes. The produced data are packetized and then enqueued into the incoming buffer on the source node, which then distributes the content according to the protocol specification. Alternatively, the application developers may opt to implement their own data sources based on the interfaces we provide.

Notably, the entire message switch module has only two threads (for *network* and *engine*, respectively), but is able to handle hundreds of concurrent connections, and switch incoming or outgoing packets immediately. To reduce context switches, we allow all nodes within each emulation entity to multiplex a single message switch in a weighted round-robin fashion, with dynamically tunable weights. Each entity runs only one *engine* and one *network*, but is able to efficiently manage the connections and buffers of hundreds of nodes with minimal CPU usage and memory footprint. Beside the routine task of switching packets and emulating bandwidth, it also has a set of built-in utilities including timers, throughput measurement, connection exception handling, etc.

2.4. The mirror scheme

Two critical problems need to be handled for a distributed architecture. First, implementing the MAC

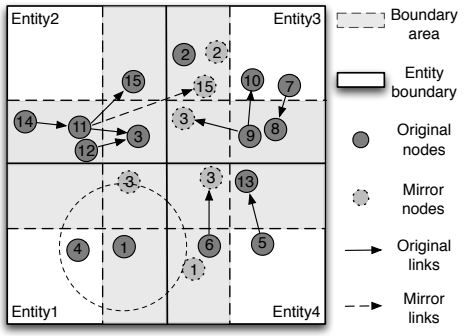

Figure 3. A typical topology in *Drift*, with example mirror node and mirror links. In this example, all nodes have uniform transmission and interference range. For non-uniform cases, the boundary area has to be specified for each node.

layer model in a distributed manner requires each node to be aware of the activities of its neighbors, even if the neighbors are located in a different entity (*i.e.*, a different process). Second, a distributed architecture requires seamless migration of nodes from one entity to another, i.e., nodes' identifications and status are kept intact even after its migration. More importantly, it requires that a node does not have to reset the existing TCP connections, even after it migrates to another entity and accordingly changes its IP and port number.

To tackle the above problems, we design a novel scheme called *mirror*. Simply put, we create copies of the original node inside neighboring entities, when it has the tendency of migration. For ease of exposition, we introduce the notation of *boundary area*, i.e., the set of locations whose distances to the edge of the geographical area represented by an entity are smaller than the transmission range (Fig. 3). Once a node moves into the boundary area, it sends mirror requests to neighboring entities (3 at most, as is the case of node 3 in Fig. 3). Based on the information piggybacked in the request message, each neighboring entity creates a mirror with the same properties (ID, location, clock, etc.) as the original node. To save computation cost, mirror nodes do not run their own mobility models. Rather, their locations are updated according to the information periodically sent from the original node. To keep the continuity of TCP/UDP connections, we set up *mirror links* immediately after the birth of mirror nodes, i.e., all connections with the original nodes are cloned for the mirrors and vice versa.

While a node resides in the boundary, its transmission and interference range span over 2 or 4 entities. At this time, all versions (one original node plus 1 or 3 mirrors) of the node periodically exchange channel information, so as to obtain a consistent view of the neighborhood's channel status. The frequency of such exchanges is a tunable parameter that varies according to the raw stability of throughput. Once the original node moves out of the boundary area, the version corresponding to its current location assumes its role, while all other versions are deleted. With such a *mirror* scheme, all nodes in *Drift* become localized and autonomous, except those in the boundary area, which can be an arbitrarily small portion as we reduce the ratio of boundary area to topology size.

2.5. Other salient features

The basic infrastructure of *Drift* involves approximately 18,000 lines of code in C++. It can be compiled on all UNIX variants without the tedious process of configuring system kernel. It provides simple configuration files with prescribed format so that an experiment can be rapidly set up with a server cluster or even several commodity PCs on hand. It also supports interactive emulation with an application called *Observer*, through which users can query, monitor and control the emulated nodes at run-time in a graphical user interface.

In *Drift*, an application is considered as a "message processor" that handles incoming packets according to a certain protocol specification. In general, developing new applications only involves modifications to a *message processor function* (which lies in the *algorithm* layer). For instance, an application layer routing protocol can be prototyped by dictating the responses of each node upon route discovery messages. After a route is established, incoming data packets are forwarded to the best nexthop by calling a *send function*, which passes the processed packets to lower layers for transmission.

3. Performance Evaluation

With the above design choices, we believe *Drift* is able to achieve much higher performance than contemporary wireless network emulators. In this section, we describe our experimentations on quantifying the scalability and accuracy of *Drift*.

3.1. Scalability

We demonstrate *Drift*'s scalability in two aspects: *capacity*, in terms of the number of nodes or the amount of traffic it can accommodate, and *efficiency*, in terms of its usage of resources in the emulation hosts.

3.1.1 Single-host capacity and efficiency

We compare *Drift*'s capacity to the most scalable wireless emulator so far, *MobiNet* [16], within the same

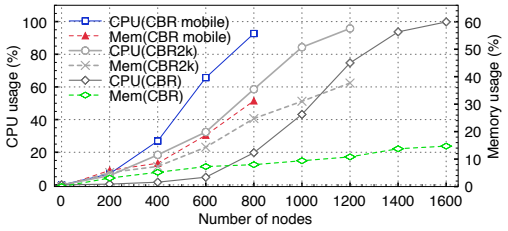

Figure 4. CPU and memory usages vs. the number of nodes in a single host.

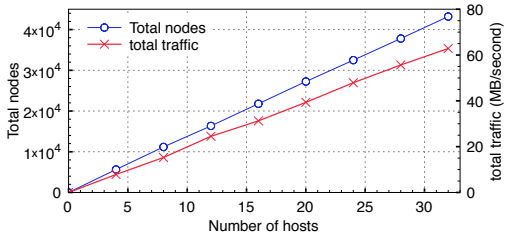

Figure 5. Node capacity and traffic capacity vs. the number of hosts.

benchmark scenario. Specifically, a total of N nodes are randomly deployed in a rectangular terrain and $\frac{N}{2}$ single-hop UDP constant bit rate (CBR) flows are started, with a rate of 256 Bytes/second and 64 Bytes packet size. The experiments are running on a 2.0GHz machine with 512MB memory, under Slackware Linux 10.2. We vary the value of N and monitor the CPU and memory usage. We observe that *Drift* is able to support up to 1600 nodes in a single machine (Fig. 4). With the same application and 3 machines of the same hardware configuration, however, *MobiNet* can only support 200 nodes at its full capacity [16]. This is because *Drift* is a lightweight application layer emulation framework, while *MobiNet*, as most existing wireless emulators, resides in the kernel level. More importantly, all pairwise flows in *MobiNet* must be routed through the central controller which enforces bandwidth and delay constraints, while *Drift* achieves the same objective using a single local host and with efficient message switches shared by hundreds of nodes. Note that the experiment with *Drift* is based on the shadowing model and the abstract MAC model. With the analytical MAC, the throughput of each link only needs to be computed once (since the flow structure is static), thus the CPU load will be further reduced.

The above experiment is designed in accordance to [16], so as to obtain a fair comparison between *Drift* and *MobiNet*. Note that the node capacity and traffic capacity are closely related. With lighter traffic, *Drift* is able to support much more nodes. On the other hand, the node capacity of *Drift* will decrease when the traffic is heavy, and mobility is turned on. In Fig. 4, we also demonstrate a scenario (CBR2k) with much heavier traffic (2KB/second) than the above CBR application. In this case, the node capacity of *Drift* reduces to around 1200. In another scenario (CBRmobile), each node moves according to the random waypoint model with a mean speed of 2 m/s while maintaining 2KB/second CBR session with a randomly selected neighbor. As shown in Fig. 4, the maximum node capacity is further reduced to around 800. Note that at a large scale, CPU becomes the bottleneck, hence we only focus on CPU usage in the following tests.

3.1.2 Benefits of distributed execution

The capacity and efficiency of *Drift* are best illustrated in distributed execution — an emulation experiment in *Drift* can be distributed to multiple entities if the maximum number of open sockets may exceed the kernel limit on per-process socket set size; and to multiple hosts if its CPU and memory consumption may exceed the capacity of a single host.

As a demonstration, we examine the node capacity of *Drift* as a function of the number of emulation hosts. We increase the number of nodes while keeping roughly constant node density (4 neighbors per node). Each node runs the neighbor discovery, *i.e.*, a "hello" message is broadcast to all neighbors every second, and a response is sent back upon receiving the message. All nodes are moving according to the random waypoint model, with a mean speed of 2 m/s. We record the total number of nodes when the CPU usage of each emulation host reaches 100% (Fig. 5). With 32 2GHz hosts, Drift is able to scale to a total of $43,200$ nodes! More importantly, *Drift*'s node capacity increases linearly as more hosts are added. The slope of the line (*i.e.*, per-host capacity) may vary depending on the computation and communication intensity of the application protocol.

To test the traffic capacity of the distributed architecture, we fix the number of nodes per-host to 500, and let half of them transmit text files to a randomly selected neighbor at a constant rate through TCP connection. We measure the achievable throughput (the aggregate throughput when CPU usage reaches 100%) while increasing the number of emulation hosts. Again, we observe linear increase of the traffic as a function of the number of hosts (Fig. 5). With 32 hosts, *Drift* is able to accommodate more than 60 MBytes/second traffic. In effect, the capacity of *Drift* can keep increasing, as long as the aggregate throughput of the target network does not exceed the capacity of the network switch that connects all emulation hosts. By contrast, no matter how many emulation hosts are used, MobiNet's capacity is upper-bounded at 5.7 MB/s [16], which is exactly the traffic capacity of the central controller. Such traffic capacity is insufficient for large topologies with thousands of links and several hundred KB/s traffic per-link.

Time	Event
7.7	Source14 starts unicast to Node3 and Node15.
24.1	Source12 starts unicast to Node3.
43.9	Node3 moves into the area of Entity3, and disconnects with Source11 and Source12.
67.1	Source9 starts unicast to Node3 and Node10.
96.5	Source7 starts unicast to Node8.
118.5	Source6 starts unicast to Node3.
120.0	Node3 moves into the area of Entity4. Source5 starts unicast to Node13.
137.3	Node3 moves to the bottom-right corner. All connections with it are broken. The original Node3 and the two mirrors are removed.

Table 1. Accuracy test scenario for the abstract MAC

3.2. Accuracy

To evaluate the accuracy of *Drift*, we focus on its MAC and physical layers since they are based on simulated models.

3.2.1 MAC layer accuracy

Due to the difficulties of implementing the details of a MAC protocol (Sec. 2.2.1), *the accuracy of the simulated MAC models in an emulator is upper-bounded by the accuracy of a packet level discrete-event simulator*. Therefore, we compare the abstract and analytical MAC models in *Drift* with a highly detailed simulator, *Qualnet* [11], which is known to be able to capture the packet level behavior of a real MAC. Specifically, we compare with the 802.11b MAC in *Qualnet 4.0* under its default settings. To highlight the MAC layer functionalities, we adopt UDP CBR transmissions with static routing, assuming the PHY link reception probability is 1, so that throughput only depends on the competition among different links.

Accuracy of the abstract model. we consider the case where node 3 in Fig. 3 roams around, and the flows in its neighborhood vary over time (Table. 3.2.1). Fig. 6 plots the throughput variation of all versions (original node and mirrors) of the mobile node 3, together with the corresponding *Qualnet* simulation results. Throughout the data session, *Drift* produces bandwidth dynamics consistent with the detailed *Qualnet* simulation. Note that its actual throughput has overestimations because it assumes a perfect MAC protocol without any overhead. According to real world measurement, the overhead of 802.11 MAC takes up more than 25% of the overall transmission time for a single link [20]. When multiple links are contending for the channel, and the RTS/CTS are turned on (as in our experiment with Qualnet), the data throughput is even lower due to the imperfect scheduling and the control message overhead. This

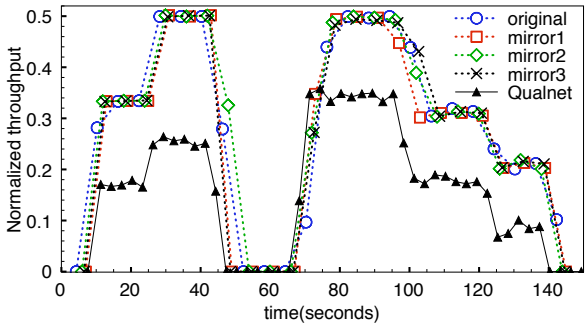

Figure 6. Throughput of a *Drift* mobile node compared with *Qualnet*.

explains the gap between the detailed MAC model in *Qualnet* and the abstract model in *Drift*.

Recall that a node in the boundary area needs to exchange channel status information periodically with its mirror versions. A natural question might be whether such interactions cause inconsistencies between different versions of a node. From Fig. 6, we also observe that all 4 versions of the node show consistent behavior, indicated by throughput. Here we intentionally set the information update period to a large value (6 seconds). With more frequent interactions, different versions of a node show more consistent behavior with each other. We note that for understanding the general variation of throughput, there is no need to perfectly synchronize the original nodes and mirrors. Without going into the microsecond level details, we can still well observe the general variation of throughput due to network dynamics such as mobility.

Accuracy of the analytical model. We evaluate the accuracy of the analytical model on a grid topology shown in Fig. 7, where each node sets up a CBR session with a randomly selected neighbor. We use the default parameters in the Qualnet MAC protocol and initialize the analytical MAC with the same parameters. Since the analytical model accurately identifies hidden terminal, exposed terminal, and other flow starvation effects in an 802.11-like MAC protocol [8], the resulting per-link

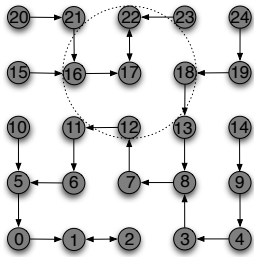

Figure 7. A grid topology with random data sessions.

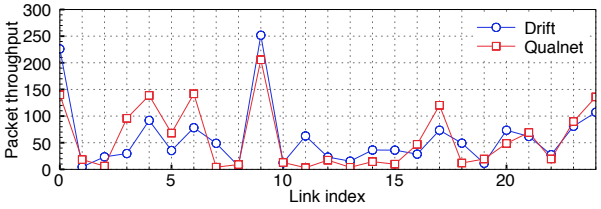

Figure 8. Per-link throughput (packets/sec.) of the grid topology (CBR sessions, 11Mb/s data rate, 1KB packet).

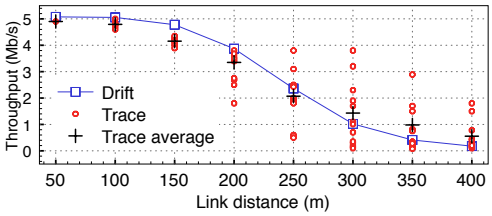

Figure 9. Link layer throughput comparison between *Drift* and real-world measurement.

throughput can approximate that of the detailed simulation by *Qualnet* (Fig. 8). Though it is more accurate than the abstract MAC, the analytical MAC has limited flexibility, *i.e.*, it cannot be applied to highly dynamic scenarios where the node moves with high speed and the flow structure varies at a time scale less than tens of seconds.

3.2.2 Accuracy of the physical layer model

We use the PHY model in *Drift* to predict the link layer throughput subject to path-loss effects. Specifically, the input to *Drift* is the MAC layer settings and the empirical physical layer characteristics (σ, β and T) measured in [3]. The output is the average link layer throughput (when transmitting UDP data) for different distances. As a benchmark comparison, we take the average RSS at each link distance in [3] (Section "backhaul link measurement"), and trace the throughput of different links with the same RSS. Fig. 9 shows the link layer throughput of *Drift* in comparison with the measurement traces sampled every 50 meters. Due to the site-specific nature of the wireless environment, it is infeasible to accurately determine the throughput of a link based on link distance. However, the analytical PHY model in *Drift* is still able to predict the general trend of throughput degradation caused by large-scale path-loss effect, *i.e.*, the shadowing effect.

3.2.3 Accuracy of bandwidth enforcement

Conventional wisdom has mostly focused on emulating wireless network stacks at the kernel level of the wireline hosts [16, 21, 22]. Traffic is routed through a central host that enforces bandwidth constraint of the corresponding link. In contrast, the bandwidth constraint in *Drift* is enforced by the application level message switch, which is decentralized and much more lightweight. One may wonder whether such a scheme can accurately perform bandwidth emulation. Our experiments have validated that the emulation error in *Drift* is within 0.1%, as long as the required bandwidth does not exceed the traffic capacity. We omit details of the experiments due to space constraints.

4. Related Work

Most existing wireless network emulators are based on a centralized architecture. In MobiEmu [21], for example, emulation hosts send packets to a dedicated central controller, which determines packet loss, delay, node mobility and other network dynamics using simulation. MobiNet [16] adopts a similar mechanism, but adds more network models including simple physical layer, MAC and mobility models. Since all the computations required by network models have to be done at the central controller, and all traffic has to be routed through it, such an approach restricts its scalability to the capacity of a single host, i.e., the central controller.

An alternative approach is trace-based emulation (see, *e.g.,* [17]), which tries to replay real-world scenarios based on the traces from existing wireless networks. However, it is suitable for only end-to-end protocol evaluation, rather than testing large-scale multi-hop wireless networks, whose traces are hard to obtain and replay.

Some emerging emulation platforms feature the integration of emulation, simulation and even physical devices to support the evaluation of wireless networks. For example, EmStar [9] uses real wireless sensors to provide the channel model for simulators. Simulated nodes are mapped to a set of sensor motes that send and receive real packets. However, the one-to-one mapping results in limited scalability. The TWINE [23] testbed goes further by incorporating the Qualnet into an emulation framework, while providing interfaces with real network devices. It takes advantage of the fine-grained simulations for MAC and physical layer protocols to ensure realism. However, with packet level granularity and the fine time synchronization managed by a dedicated controller, it only scales to around 4 wireless nodes in each emulation host.

There are a relatively large number of experiment platforms constructed by real wireless devices (see, for example, [6,13,14,18]). They can provide accurate evaluation of network protocols. However, experimenting with real wireless devices takes a considerable amount

of time, which is undesirable if protocol designers hope to try out alternatives efficiently under a variety of settings. Moreover, due to the cost of real mobile wireless nodes, it is sometimes infeasible to test networks with more than dozens of nodes, subject to funding constraints.

Simulation platforms have also been widely used by the research community. Since no real packets are transmitted, it is easier to achieve scalability in simulators, especially in distributed simulation engines [11,19]. The common limitation of simulators, as noted above, is the need for protocol re-implementation [16], when switching from a specific simulator to real operating systems. The simulated transport and IP layer protocols may also compromise the simulation results [12].

5. Conclusion

In this paper, we described *Drift*, a highly efficient emulation platform for the development and validation of application-layer algorithms in large scale wireless networks. *Drift* is designed from scratch to support scalable emulation within a cluster environment, where thousands of nodes are running in each host, and nodes can seamlessly migrate from one host to another. In order to maximize scalability with minimal loss of accuracy, *Drift* adopts abstract and analysis-based lower layer models. Our extensive evaluations have demonstrated the high performance of *Drift* in terms of scalability and accuracy. We have planned to release *Drift* as an open source platform, and as the full potential of *Drift* is explored and more researchers adopt this emulation framework, we believe it will become a valuable tool to evaluate protocols—such as network coding—at the application layer in large scale wireless networks.

References

[1] S. Biswas and R. Morris. ExOR: Opportunistic Multi-hop Routing for Wireless Networks. In *Proc. of ACM SIGCOMM*, 2005.

[2] J. L. Boudec and M. Vojnovi. Perfect Simulation and Stationarity of A Class of Mobility Models. In *Proc. of IEEE INFOCOM*, 2005.

[3] J. Camp, J. Robinson, C. Steger, and E. Knightly. Measurement Driven Deployment of a Two-tier Urban Mesh Access Network. In *Proc. of ACM MobiSys*, 2006.

[4] S. Chachulski, M. Jennings, S. Katti, and D. Katabi. Trading Structure for Randomness in Wireless Opportunistic Routing. In *Proc. of ACM SIGCOMM*, 2007.

[5] D. Couto, D. Aguayo, J. Bicket, and R. Morris. A High-throughput Path Metric for Multi-hop Wireless Routing. In *Proc. of ACM MobiCom*, 2003.

[6] P. De, A. Raniwala, R. Krishnan, K. Tatavarthi, J. Modi, N. A. Syed, S. Sharma, and T. Chiueh. MiNT-m: An Autonomous Mobile Wireless Experimentation Platform. In *Proc. of ACM MobiSys*, 2006.

[7] D. Ganesan, B. Krishnamachari, A. Woo, D. Culler, D. Estrin, and S. Wicker. Complex Behavior at Scale: An Experimental Study of Low-Power Wireless Sensor Networks. Technical report, UCLA, 2006.

[8] M. Garetto, T. Salonidis, and E. Knightly. Modeling Per-flow Throughput And Capturing Starvation in CSMA Multi-hop Wireless Networks. In *Proc. of IEEE INFOCOM*, 2006.

[9] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin. EmStar: a Software Environment for Developing and Deploying Wireless Sensor Networks. In *Proc. of USENIX*, 2004.

[10] J. Heidemann, N. Bulusu, J. Elson, C. Intanagonwiwat, W. Ye, D. Estrin, and R. Govindan. Effects of Detail in Wireless Network Simulation. In *Proc. of SCS Multiconference on Distributed Simulation*, 2001.

[11] S. N. T. Inc. Qualnet. http://www.scalable-networks.com/.

[12] S. Jansen and A. McGregor. Simulation With Real world Network Stacks. In *Proc. of WSC*, 2005.

[13] D. Johnson, T. Stack, R. Fish, D. Flickinger, L. Stoller, R. Ricci, and J. Lepreau. Mobile Emulab: A Robotic Wireless and Sensor Network Testbed. In *Proc. of IEEE INFOCOM*, 2006.

[14] G. Judd and P. Steenkiste. Using emulation to understand and improve wireless networks and applications. In *Proc. of NSDI*, 2005.

[15] V. S. A. Kumar, M. V. Marathe, S. Parthasarathy, and A. Srinivasan. Algorithmic Aspects of Capacity in Wireless Networks. *SIGMETRICS Perform. Eval. Rev.*, 33(1), 2005.

[16] P. Mahadevan, A. Rodriguez, D. Becker, and A. Vahdat. MobiNet: A Scalable Emulation Infrastructure for Ad Hoc and Wireless Networks. *Mobile Computing and Communications Review*, 10(2), 2006.

[17] B. Noble, M. Satyanarayanan, G. Nguyen, and R. H. Katz. Trace-based Mobile Network Emulation. In *Proc. of ACM SIGCOMM*, 1997.

[18] K. Ramachandran, S. Kaul, S. Mathur, M. Gruteser, and I. Seskar. Towards Large-scale Mobile Network Emulation Through Spatial Switching on a Wireless Grid. In *Proc. of SIGCOMM EWIND*, 2005.

[19] G. F. Riley. Large-Scale Network Simulations with GTNetS. In *Proc. of Winter Simulation Conference (WSC)*, 2003.

[20] M. Rodrig, C. Reis, R. Mahajan, D. Wetherall, and J. Zahorjan. Measurement-based Characterization of 802.11 in a Hotspot Setting. In *Proc. of ACM SIGCOMM E-WIND*, 2005.

[21] Y. Zhang and W. Li. An Integrated Environment for Testing Mobile Ad-hoc Networks. In *Proc. of ACM MobiHoc*, 2002.

[22] P. Zheng and L. M. Ni. EMPOWER: A Network Emulator for Wireline and Wireless Networks. In *Proc. of IEEE INFOCOM*, 2003.

[23] J. Zhou, Z. Ji, and R. Bagrodia. TWINE: A Hybrid Emulation Testbed for Wireless Networks and Applications. In *Proc. of INFOCOM*, 2006.