Apples, Oranges, and Testbeds

Koen Langendoen
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology, The Netherlands
K.G.Langendoen@tudelft.nl

Abstract— **Research into wireless sensor networks is rapidly moving from simulations to realistic testbeds. The widely varying characteristics (e.g., radio hardware, #nodes, topology) of various testbeds raises concerns about the validity of results across different testbeds. This paper presents empirical data of an experiment involving one application (Surge), two routing protocols (MultiHop and MintRoute), and two testbeds (MoteLab and MistLab). The outcome is somewhat mixed. When increasing the data rate, congestion causes goodput to fall off in a similar fashion on both testbeds, which is good, but only when ignoring MintRoute ill-behaving for low rates on MoteLab, which is bad. Accounting for differences in communication hardware is necessary, but even then results should be taken with a grain of salt. This certainly holds for TOSSIM simulation results, since we found that they generally do not match those of physical testbeds.**

I. Introduction

Research into wireless sensor networks is rapidly maturing as can be witnessed from the explosive growth in number of testbeds. Every respected group now has its own, and some testbeds have even been been made publicly available so anybody can – and is expected to – prototype algorithms and protocols. The shift from simulation-based research to testbed measurements raises important questions about the validity of the results obtained on individual testbeds. Do performance results from protocol X on testbed Y also hold for testbed Z? Certainly not! Differences in type and number of nodes, network topology, and link quality guarantee that every testbed is unique. Worse, even results from a single testbed are difficult, if not impossible, to reproduce exactly over time (i.e. one week later) due to the inherent instability of wireless links and the inevitable node failures (and resurrections).

These concerns make interpreting results from different testbeds look like comparing apples and oranges. Fortunately, the situation is not as bleak as that. The problems with link quality fluctuations are generally known to researchers, and therefore most protocols and algorithms tolerate a certain degree of variation in performance of the wireless network. Nevertheless, many protocols do break down when operated outside their original settings as many have found out when porting code from one platform to the other, for example, when trying to run third-party TinyOS software on their own testbed.

This paper presents empirical data of an experiment involving one application (Surge), two routing protocols (MultiHop and MintRoute), and two testbeds (MoteLab and MistLab). The outcome is somewhat mixed. When increasing the data rate, congestion causes goodput to fall off in a similar fashion on both testbeds, which is good, but only when ignoring MintRoute ill-behaving for low rates on MoteLab, which is bad. Accounting for differences in communication hardware (CC1000 vs. CC2420 radios) is necessary, but even then results should be taken with a grain of salt. For example, generally MultiHop outperforms MintRoute by a small margin, but under specific circumstances the reverse holds! Also, the TOSSIM simulation results included for reference do not compare to those obtained on the physical testbeds.

Although protocol and algorithm comparisons on a single sensor network testbed are quite common, to the best of our knowledge, this is the first paper to compare results across multiple testbeds. The remainder of this paper is structured as follows. Section II briefly reviews related work, and describes the methodology used in our comparison experiments. Section III discusses the setup of the experiments, and various results are presented in Section IV. The relevance of these results is discussed in Section V, and Section VI concludes the paper.

II. Methodology

Wireless communications are notoriously difficult to characterize because of physical effects like fading, multi-path reflections, interference, and antenna diversity causing irregular reception patterns. In the case of sensor networks, several studies have demonstrated severe performance effects across a considerable fraction of the transmission range, the so-called gray area, for different types of radios and environments [7, 10, 11, 12]. This makes it hard to build testbeds that allow for controllable experiments with repeatable results. Although the use of simple, low-cost radios aggravates problems, wireless testbeds including more advanced radios (IEEE 802.11) generally score bad on repeatability as well, as observed in a recent survey by De et al. [1]. Some researchers therefore resort to exotic solutions involving remotely controlled equipment in an anechoic chamber [8]. Besides being costly, such an approach is unlikely to deliver results that are of use in a practical setting.

To provide the research community with a reference point on how good (or bad) WSN testbeds perform relatively to each other, we need a benchmark that captures various metrics of interest. Exactly which metrics to consider is an open question, since only a few papers address benchmarking of sensor networks. TinyBench [2] puts out a first proposal for a standardized benchmark suite focusing on single node performance parameters (code size, runtime and power consumption). Bisque [5], a benchmark for in-network query processing, considers multiple nodes (arranged in a square grid topology) and reports power consumption, response time, and relative error rate (accuracy) of individual queries. It is left unspecified how to normalize these (application-specific) metrics to account for differences in processing and communication speed of various testbeds. The lack of common metrics has prompted us to use our own. One set for characterizing individual testbeds, and another capturing application performance to allow for comparisons *between* testbeds.

To collect basic information characterizing individual testbeds we want to record the number of nodes, their topology, and the link qualities between them. To limit the amount of data in the light of changing link conditions, and hence topology, we propose to capture the essentials only during a benchmark run:

TABLE I
Testbed characterization parameters.

parameter	description
#nodes	number of active nodes
connectivity	average number of neighbors over all active nodes (topology information)
link capacity	maximum raw radio throughput over any one link [bytes/s]
link quality	raw packet success rate (no retransmissions) averaged over all links

By logging these parameters over various runs, we can determine important second order parameters like stability and drift. Note that we do *not* include any power consumption numbers; although the focus on energy efficiency is a defining characteristic of WSN research, none of the testbeds that we are aware of offers the capability to measure power consumption of applications running on multiple nodes.

To allow for cross-testbed comparisons of application performance we need to monitor a set of application-level parameters. However, to understand the impact of testbed characteristics on application behavior, we also need to monitor a set of parameters about the system-level software that drives the raw testbed hardware. The choice of parameters is vast, so we must limit ourselves to some manageable set. In this paper we focus on a small set of communication-related parameters, since generally sensor nodes spend most of their energy on driving the radio.

At the application-level we monitor end-to-end goodput for the prototypical convergecast communication pattern embodied in many monitoring applications (e.g., Surge) where all nodes periodically send a message to the sink node in the network. By varying the message injection period we can control the communication load, and study how the protocol stack copes with it, i.e. how goodput

TABLE II
COMMUNICATION-RELATED PARAMETERS.

parameter	description
goodput	fraction of injected messages delivered at the sink
hop count	average hop count of the delivered messages
tree depth	average hop count over all paths to the sink
route quality	one-hop message success rate averaged over all used links

falls off when increasing the send rate. Another parameter of interest is the average path length (hop count), which is a metric for the induced aggregate load on the network (assuming no retransmissions) and captures changes in fairness between near and far nodes when the communication load varies.

For the routing component, which is responsible for setting up a spanning tree to the sink, we monitor how it trades off link quality and path length (short paths with long, poor links vs. long paths with short, good links). In particular, we monitor tree depth and the success rate of raw one-hop (intermediate) transmissions. These parameters will change when increasing the communication load, because the underlying MAC protocol will at some point break down and "lose" messages due to collisions and/or buffer overflows.

By monitoring parameters at different layers in the protocol stack we can perform a detailed analysis of the observed application behavior in contrast to typical black-box approaches that only measure end-to-end performance parameters (goodput). The work by Malesci et al. is a fine example of the latter approach and closely resembles the setup in this paper (Surge + different protocol stacks), but limits itself to a single testbed [6].

III. SETUP

The comparison experiments were carried over a time period of about two months (January 27 - April 12, 2006). The setup of these experiments involved two testbeds, the accompanying platform-specific TinyOS protocols, two routing protocols, and a modified Surge application. All of which will be discussed below.

A. Testbeds

When deciding on which testbeds to use, we initially looked for publicly accessible testbeds with similar node hardware to ease comparisons and avoid having to account for differences in communication speeds, etc. Of course, life is not that easy, so we ended up with two testbeds, MistLab from MIT [13] and MoteLab from Harvard [9], and three node types (mica2, micaz, and Tmote Sky). MistLab consists of a mixture of 47 mica2 nodes and 14 Cricket nodes spread across multiple rooms located on the 9th floor of MIT's CS department. For our experiments we only used the popular mica2 nodes from Crossbow, which are equipped with a CC1000 radio from Chipcon. The CC1000 radio provides a raw speed of 19.2 kbps, but under TinyOS (see below) the effective link capacity is 783 bytes/s (see Table III).

When we started using MoteLab it was configured as a 30-node testbed equipped with micaz nodes, also manufactured by Crossbow. In contrast to the mica2 nodes, the micaz nodes are equipped with a packet-based CC2420 radio. The maximum throughput available to TinyOS applications (without routing) is about 3 KB/s. In the middle of February MoteLab was upgraded to host an impressive number of 190 Tmote Sky nodes manufactured by Moteiv. Keeping so many nodes functioning, however, is quite difficult and during our experiments we were only able to operate (at most) 76 nodes (see Table III). The Tmote Sky nodes are equipped with the same CC2420 radio as the micaz nodes, but since they are operated on the same three floors of Harvard's EECS building the node density is a lot higher; on average a Tmote Sky node has about twice as many neighbors in range as a micaz node.

Since much research is still performed in simulation we included the popular TOSSIM [4] framework, which is generally used for prototyping TinyOS applications. We used the default configuration that models an old-style mica node (RFM 1000 radio), because the optional mica2 protocol stack (CC1000 radio) yielded unexplainable results not matching any physical testbed observations. To limit simulation times we set the TOSSIM parameters to model 30 nodes randomly placed in

TABLE III

TESTBED CHARACTERISTICS (MIN - AVG - MAX).

	MistLab	MoteLab		TOSSIM
parameter	mica2	micaz	Tmote Sky	mica
#nodes	30 - 35.3 - 39	12 - 23.7 - 27	27 - 50.9 - 76	30
connectivity	9.6 - 11.8 - 14.2	4.6 - 5.4 - 6.3	4.4 - 10.4 - 17.5	6.9 - 8.1 - 9.8
link capacity [KB/s]	0.76	3.12	3.83	1.33
link quality	0.74 - 0.77 - 0.84	–	0.74 - 0.82 - 0.89	0.45 - 0.52 - 0.58

a 20x20 grid with a radio range of 5 units. We used the lossy radio model provided with TOSSIM (`lossy-20x20-5.nss`), where bit errors depend on the distance from sender to receiver and background noise (other nodes communicating). Experiments were repeated with different topologies and random seeds, explaining the variation in connectivity and link qualities (see Table III).

The characteristics of the three physical testbeds under study and the TOSSIM simulator are concisely represented in Table III. Most parameters show a considerable spread, which is for a large part caused by changes in the configuration of the testbeds. Figure 1 shows the number of active nodes in each testbed (Mistlab, Motelab-z, and Motelab-sky) over the course of the experiments, and Figure 2 shows the corresponding changes in connectivity. Note that there is no direct correlation between node failures and connectivity; in the case of Motelab-z a significant reduction in nodes (at the end of its lifetime) has little effect on connectivity, but for Motelab-sky node failures correspond with a reduction in connectivity. We conjecture that these differences are caused by random failures (lower connectivity for everybody) vs. localized failures (no change for most survivors).

We have tried to determine what other factors besides testbed configuration contribute to the observed variability in characterization parameters, but with no success. For example, we have compared measurements taken during the day with measurements at night to see if human activity influences link quality, but no significant differences were found, ruling out its importance. The instability of the testbeds made it impossible to study long-term effects like drift. As far as repeatability is concerned, we observed that back-to-back measurements generally yield similar characteristics.

B. TinyOS and friends

The software used in the comparison experiments is based on the reference TinyOS implementation (snapshot release 1.1.15) available at the SourceForge CVS repository as of March 13, 2006. We have been using the standard protocol stacks for the hardware platforms in the two testbeds under study. For the mica2 platform (MistLab) the CC1000 radio is driven by a MAC layer that, in principle, uses low-power listening to save energy. Since we do not monitor energy efficiency, we simply use the default setting of always keeping the radio on. The MAC layer itself does not perform any retransmissions, but notifies the routing layer above of missing acknowledgements for unicast traffic.

Although the CC2420 radio used in both MoteLab setups (micaz and Tmote Sky) is IEEE 802.15.4 compliant, the built-in MAC functionality is *not* used. In both cases the TinyOS protocol stack only uses the standardized framing structure (packet layout), while implementing the medium access control in software, much like for the mica2 platform (without low-power listening).

The next level up in the protocol stack is the routing layer, which in TinyOS takes care of sending messages from individual nodes to the sink node along paths that may span multiple hops. We have been experimenting with two implementations:

MultiHop This protocol selects routes based on the shortest hop count to the sink. We made a few alterations to the code in `/lib/Route`. First, we changed the (hard-coded) identity of the sink node from 0 to 1 since none of the physical testbeds includes a node zero. Second, we set the default route update interval to 20 seconds to control the overhead

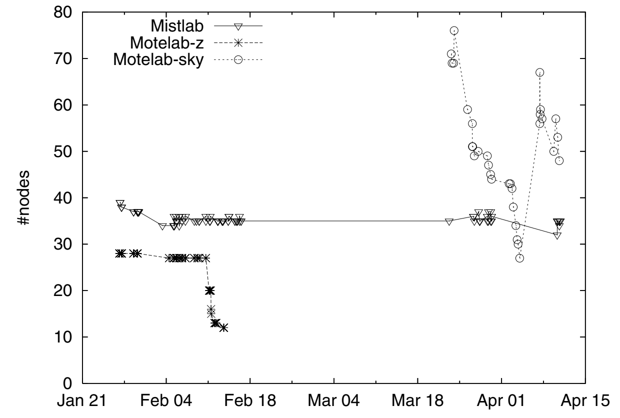

Fig. 1. Number of working nodes over time.

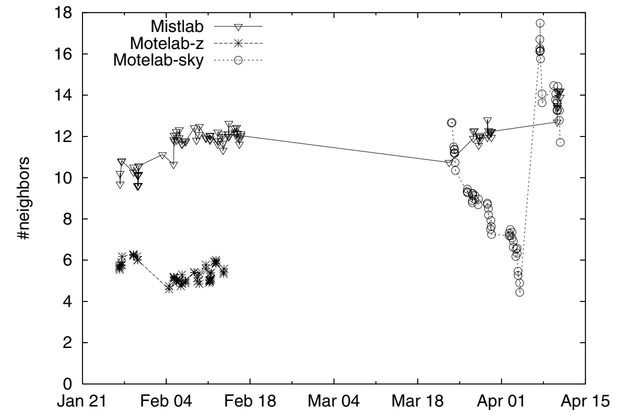

Fig. 2. Connectivity over time.

for route setup and maintenance. Third, we changed the default destination, used when the parent in the spanning tree is unknown, from `TOS_BCAST_ADDR` to `BASE_STATION_ADDRESS` allowing all nodes within a one-hop distance from the sink to deliver their messages instead of them being silently discarded because of the destination address not matching the local address.

MintRoute This protocol considers link quality when selecting parents in the spanning tree favoring multiple hops with good links over a single, long hop with a poor link (which MultiHop would take). MintRoute's definition of link quality is not simply packet success rate (as used for testbed characterization in Table I) and the selection process also considers some other factors, see [11] for details. We applied the same set of alterations to the code in `/lib/MintRoute` as for MultiHop. However, since MintRoute is very slow in setting up routes we set the routing update interval to 10 seconds. To speed up the selection process even further, we added some skew proportional to the node number before broadcasting a route update to reduce the number of collisions. (As a side effect, the quality of the selection process was also improved.) Despite these changes, MintRoute in general takes much longer than MultiHop to set up a spanning tree to the sink.

C. *MultiRateSurge*

Surge is one of the few applications generally used to evaluate sensor networks. Nodes periodically send a message to the sink reporting a reading of their light sensor. The interval period between consecutive measurements can be controlled, allowing for experimentation with different network loads. We have adapted Surge to cycle through a pattern of decreasing intervals (3 s - 50 ms). The length of each stage is inversely proportional to the length of its inter-message period, which ensures that approximately the same number of messages is injected into the network for each stage. To avoid stages partly overlapping each other, and to allow for the underlying routing protocol to recover from heavy traffic resulting in the loss of its spanning tree, stages are separated by one minute of "silence". Figure 3 shows an example taken from the MistLab testbed displaying application-level goodput; the alternation of (shortening) active and passive periods is clearly visible. The drop off in goodput signals that the network suffers from collisions for high loads. The goodput numbers presented in Figure 3 and Section IV are obtained with an optimization to Surge that skews message injection times, as for the routing updates, to avoid all nodes sending at the same time causing collisions even for stages with long inter-message intervals.

Surge has also been adapted to obtain both the testbed characterization parameters (Table I) and

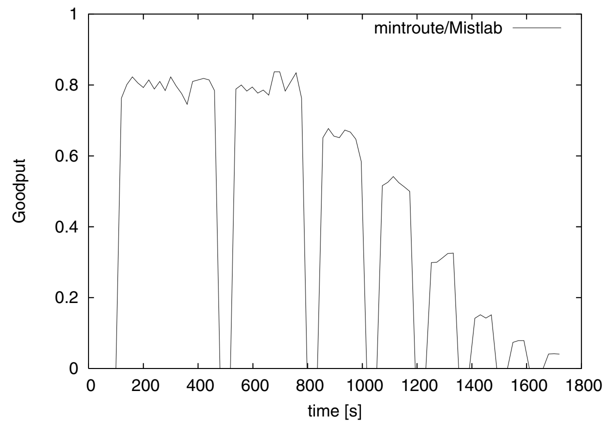

Fig. 3. Surge example run (36 nodes, 2006-02-15 18:01:13).

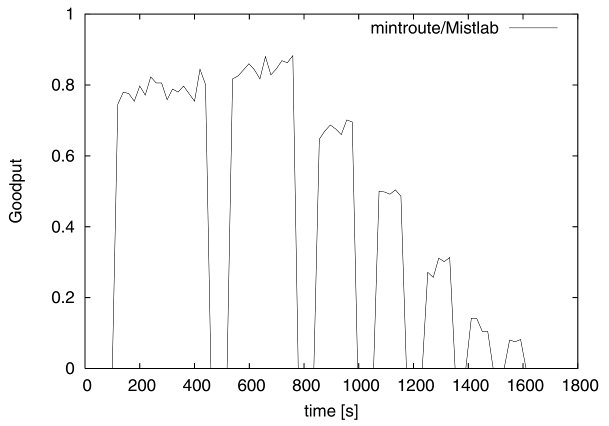

Fig. 4. Surge/MintRoute (36 nodes, 2006-02-15 18:32:07).

the communication-related parameters (Table II) with each run. To this end the routing protocols were used in promiscuous mode delivering *all* messages to the application for inspection. By peeking at the source and sequence-number fields in the routing header, which is the same for MultiHop and MintRoute, we can count the number of transmitted and lost messages from each neighboring node (i.e., parent and child nodes, as well as other nodes within radio range). From these counts, nodes compute the connectivity and link-quality parameters, which are send out over the serial port every 20 seconds. Serial output is automatically captured by the testbed software and logged for off-line analysis.

Surge messages arriving at the sink are logged through the serial port to compute communication parameters (goodput, hop count, etc.). We changed the message format to include an application-level sequence number (for computing goodput) and a hop count, which is incremented by Surge at intermediate nodes through the promiscuous routing `Intercept` interface. This interface only reports messages from child nodes (the `Snoop` interface catches messages from the other nodes), which makes it easy to compute the one-hop message success rate for the child links. These rates are logged through the serial port, and later averaged over all nodes to arrive at the route-quality parameter.

IV. Experiments

In this section we report on results obtained by running two flavors of MultiRateSurge (MultiHop

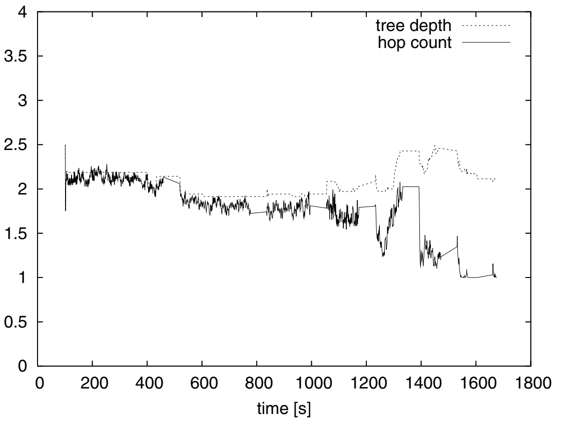

Fig. 5. MintRoute details (36 nodes, 2006-02-15 18:32:07).

and MintRoute) on the three testbeds (Mistlab, Motelab-z, and Motelab-sky), and in simulation (TOSSIM). Before getting to the main question of the paper "do experimental results hold across testbeds?" we first report on the consistency of results (over time) within a single testbed.

A. Testbed results

Figure 4 shows a second plot of the goodput results achieved with MintRoute on the MistLab testbed. This run was performed straight after the first one presented in Figure 3. Although the gooodputs obtained by these back-to-back runs are pretty similar, MintRoute did set up two different spanning trees. Small changes in message ordering and loss, cause MintRoute to derive different link qualities, occasionally resulting in the selection of another parent.

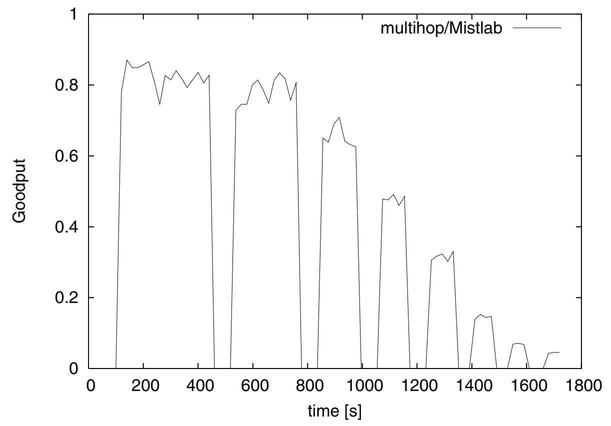

Fig. 6. Surge/MultiHop (36 nodes, 2006-02-15 20:01:01).

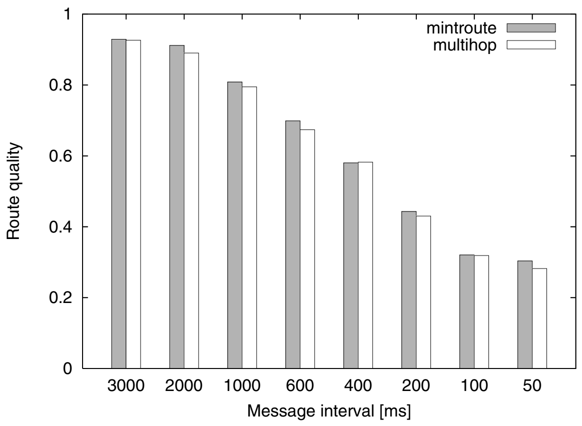

Fig. 7. Route quality comparison (cf. Fig. 4 and Fig. 6).

During the course of execution MintRoute adapts the spanning tree when link qualities diverge too much. In our case the increased load that Surge injects into the network causes an increase in the number of collisions, hence, link quality degrades forcing MintRoute to adapt the tree. This effect is shown in Figure 5, where the average tree depth increases when the message rate goes up in subsequent active periods. A consequence of the larger tree depth is that leaf nodes have less chance of seeing their messages delivered at the sink, because of the (significant) chance of messages getting lost at each additional hop. That explains why the average hop count of the messages received at the sink drops. At the highest message rate, the hop count has dropped to nearly one, showing that only the direct neighbors succeed in delivering data at the sink.

Intuition says that MintRoute, taking link quality into account, should do better than MultiHop basing its route selection solely on hop counts. Figure 6 shows the goodput for Surge with MultiHop routing on the MistLab testbed one hour later than the MintRoute experiment from Figure 4. MultiHop does remarkably well and achieves a comparable goodput profile as MintRoute. This is confirmed by the route quality metric shown in Figure 7; the links selected by MultiHop are generally as good, or as bad, as those of MintRoute. We cannot fully account for this unexpected result, but we believe it is caused by the binary distribution of link qualities for low-cost radios. Most links are either good (success rate $> 85\,\%$) or really bad (success rate $< 15\,\%$) according to [7], which deprives MultiHop from the possibility of selecting an incorrect link, since route updates simply do not make it across bad links.

Both MintRoute and MultiHop are capable of selecting good links, which explains the high goodput numbers given the relatively low raw link quality, which is on average 0.77 for the MistLab testbed (Table III). Statistical analysis of the goodput numbers for each inter-message interval for all runs over the 2 months period of experiments shows that the MistLab testbed is quite stable. For example, MultiHop's average goodput for a 3 second interval is 0.90 with a standard deviation of just 0.040 (4.5 %), and MintRoute's goodput is $0.82 \pm 6.4\,\%$ on average. For brevity we do not show individual results from the two MoteLab testbeds, but the repeatability on these testbeds was generally much lower than for MistLab. For MoteLab equipped with Tmote Sky nodes, the number of active nodes varied too much to perform a sound analysis, but for the micaz nodes we observed standard deviations of 20 % and more.

B. Comparison

Overall, counter to intuition, MultiHop outperforms MintRoute by a small margin on the MistLab testbed. To see if this result carries over to the other testbeds, we have computed for each testbed, routing protocol, and Surge interval the peak goodput averaged over all measurements taken on that testbed. Thus we do not try to account for different configurations within a testbed, simply because it was not feasible to collect enough data on each configuration with a single run of Surge lasting 30 minutes. Figure 8 shows the resulting goodput profiles for the three testbeds, as well as for the TOSSIM simulator.

The goodput numbers for the Motelab-z testbed drop off much slower than for Mistlab when increasing the load, which is a consequence of the difference in radio speed (0.76 vs. 3.1 KB/s) of the two testbeds. MultiHop and MintRoute achieve about equal performance on both testbeds, but the variability is much higher for Motelab-z. This is difficult to explain from the testbed characteristics, which are quite stable (except for the last week of deployment when the number of nodes rapidly declined, but analysis without that last week's data yields similar goodputs and standard deviations).

The results for Motelab-sky are quite puzzling. For low message rates MintRoute performs very bad (goodput $< 50\,\%$) and loses to MultiHop. For high rates, however, MintRoute wins by a factor of two. What we gathered from the log files is that MintRoute for some, yet unknown reason is very slow in setting up an initial spanning tree. Once the tree is in place, it outperforms MultiHop by selecting links of higher qualities. The large standard deviations for Motelab-sky are most likely caused by the volatile configuration of the testbed (see Figure 1). What we do not understand is why MultiHop performance drops off much faster than for Motelab-z, which used the same radio chip. Additional research is needed to clarify this issue.

Finally, the results from the TOSSIM simulator differ dramatically from those observed on the physical testbeds. The low goodput numbers cannot be explained by a difference in radio speed (on the contrary, the RFM radio is a factor 1.7 *faster* than MistLab's CC1000 radio). The fundamental issue is the propagation model used (signal strength being inversely proportional to distance in free space), which bears little resemblance with reality (gray area effect). This is reflected in the raw link quality being much lower than for the real testbeds (see Table III) causing application-level performance to degrade. This can probably be accounted for, but the fact that TOSSIM is the only testbed having MintRoute outperform MultiHop for low loads can not.

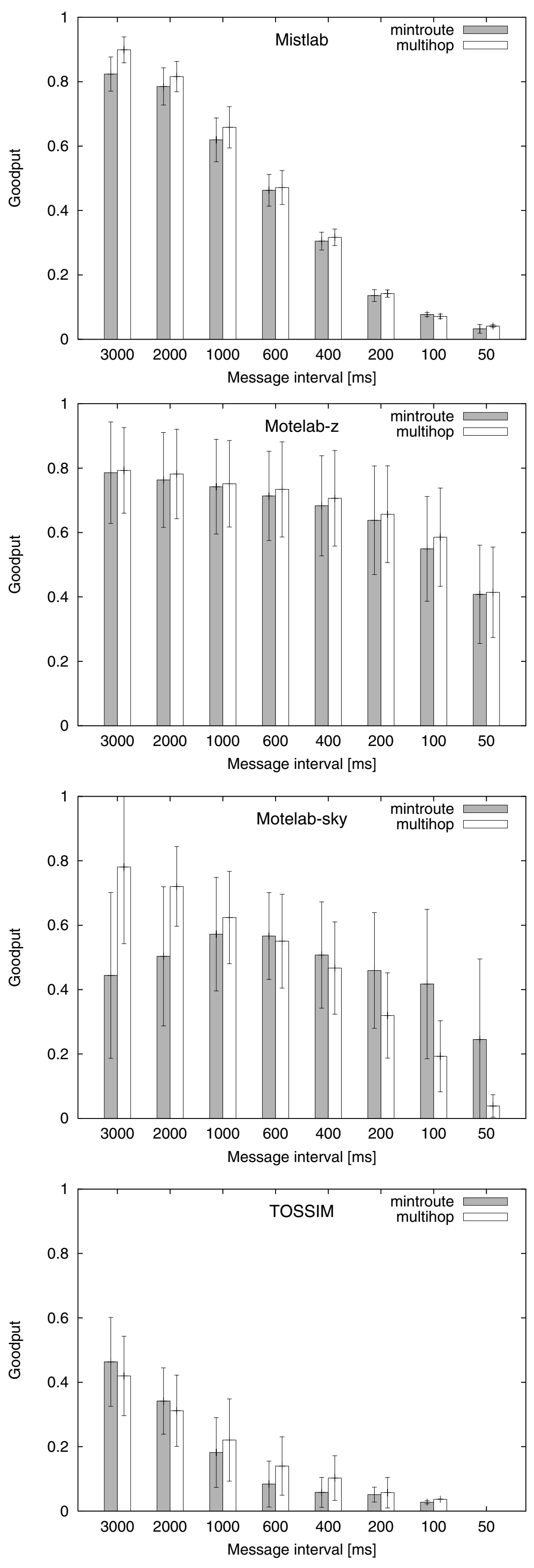

Fig. 8. Surge performance across testbeds with error bars (std.dev.).

V. Discussion

The most important observation from the Surge comparison experiment is that the three testbed profiles share little resemblance, which was to be expected, but nevertheless points out that experimental results obtained on a single testbed are very difficult to generalize. Depending on your testbed of choice MultiHop outperforms MintRoute, or vice versa. In the case of Motelab-sky, the outcome even depends on the Surge interval due to MintRoute failing to set up an initial routing tree quickly. We have observed MintRoute misbehave before [3], so we are not surprised to see it happen again. The point, however, is that only Motelab-sky triggered this behavior stressing the incompatibility between testbeds. Our experiences with TOSSIM show that using the default lossy radio model, which does not match reality, leads to different behavior at the application level. Finding a suitable radio model will be crucial for getting simulators to yield reliable results.

Despite our efforts in logging communication parameters at various levels in the protocol stack, in order to obtain a better insight in what is going on below, we frequently failed to understand Surge's performance results and could not determine who was to blame (i.e the testbed characteristics, or the routing layer?). For protocol development in general, it would be very nice if we could factor out the hardware differences as well as the environmental factors. This is not a trivial task and more work is needed in this area.

VI. Conclusions

With the shift from simulation-based research to realistic testbeds and pilot deployments of wireless sensor networks, this paper has addressed the issue of how well results translate from one platform to the other. We have taken an experimental approach and run a modified prototypical monitoring application (Surge), in combination with two routing protocols (MultiHop and MintRoute), on two publicly available testbeds (MistLab and MoteLab).

To account for hardware and environmental differences we characterize testbeds by four parameters (which vary over time): #nodes, connectivity, link capacity, and link quality. Application and routing performance is captured by another set of four parameters: goodput, hop count, tree depth, and route quality. To see how application and protocol stack respond to network load, the message injection rate was increased to the point of collapse.

Running the two flavors of Surge (MultiHop and MintRoute) on the testbeds at different times during a two month period of experiments has revealed that generally MultiHop outperforms MintRoute, but given the "right" set of conditions the reverse can be observed as well. The results for the popular TOSSIM simulator included for reference, showed that its default lossy radio model yields incompatible performance numbers with those from the physical testbeds.

In summary, comparing results from different testbeds is much like comparing apples and oranges, and simulation results should be taken with a grain of salt as well.

VII. Acknowledgements

We would like to thank the people from MoteLab (Matt Welsh and Geoff Werner-Allen) and MistLab (Kyle Jamieson) for generously providing access to their testbeds and helping out in getting software and logging to run properly and putting up with the frustrations from remote debugging with a 6 hour time difference. We also would like to thank Gertjan Halkes for helping out with interpreting testbed results during the experiments, and filling in the blanks at the very last moment.

References

[1] P. De, A. Raniwala, S. Sharma, and T.-C. Chiueh, "Design considerations for a multihop wireless network testbed," *IEEE Communications Magazine*, vol. 43, no. 10, pp. 102–109, Oct. 2005.

[2] M. Hempstead, M. Welsh, and D. Brooks, "TinyBench: The case for a standardized benchmark suite for TinyOS based wireless sensor network devices," in *29th IEEE Int. Conf. on Local Computer Networks*, Tampa, FL, Nov. 2004, pp. 585–586.

[3] K. Langendoen, A. Baggio, and O. Visser, "Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture," in *14th Int. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, Rhodes, Greece, Apr. 2006.

[4] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications," in *1st ACM Conf. on Embedded Networked Sensor Systems (SenSys 2003)*, Los Angeles, CA, Nov. 2003, pp. 126–137.

[5] Q. Luo, H. Wu, W. Xue, and B. He, “Benchmarking in-network sensor query processing,” The Hong Kong University of Science and Technology, Tech. Rep. HKUST-CS05-09, June 2005.

[6] U. Malesci and S. Madden, “A measurement-based analysis of the interaction between network layers in TinyOS,” in *3rd European Workshop on Sensor Networks (EWSN’06)*, Zurich, Switzerland, Feb. 2006, pp. 292–309.

[7] N. Reijers, G. Halkes, and K. Langendoen, “Link layer measurements in sensor networks,” in *1st IEEE Conf. on Mobile Ad-hoc and Sensor Systems (MASS 2004)*, Fort Lauderdale, FL, Oct. 2004.

[8] N. Vaidya, J. Bernhard, V. Veeravalli, P. Kumar, and R. Iyer, “Illinois wireless wind tunnel: A testbed for experimental evaluation of wireless networks,” in *Workshop on Experimental Approaches to Wireless Network Design and Analysis (E-WIND’05)*, Philadelphia, PA, Aug. 2005, pp. 64–69.

[9] G. Werner-Allen, P. Swieskowski, and M. Welsh, “MoteLab: A wireless sensor network testbed,” in *Special Track on Platform Tools and Design Methods for Network Embedded Sensors (SPOTS) associated with IPSN’05*, Los Angeles, CA, Apr. 2005.

[10] A. Willig and R. Mitschke, “Results of bit error measurements with sensor nodes and casuistic consequences for design of energy-efficient error control schemes,” in *3rd European Workshop on Sensor Networks (EWSN’06)*, Zurich, Switzerland, Feb. 2006, pp. 310–325.

[11] A. Woo, T. Tong, and D. Culler, “Taming the underlying challenges of reliable multihop routing in sensor networks,” in *1st ACM Conf. on Embedded Networked Sensor Systems (SenSys 2003)*, Los Angeles, CA, Nov. 2003, pp. 14–27.

[12] J. Zhao and R. Govindan, “Understanding packet delivery performance in dense wireless sensor networks,” in *1st ACM Conf. on Embedded Networked Sensor Systems (SenSys 2003)*, Los Angeles, CA, Nov. 2003, pp. 1–13.

[13] “Mistlab website,” http://mistlab.csail.mit.edu/.