P^3: A Practical Packet Pipeline Using Synchronous Transmissions for Wireless Sensor Networks

Manjunath Doddavenkatappa and Mun Choon Chan
School of Computing, National University of Singapore

***Abstract*—While high throughput is the key for a number of important applications of sensor networks, performance of the state-of-the-art approach is often poor in practice. This is because if even one of the channels used in its pipeline is bad, the pipeline stalls and throughput degrades significantly.**

In this paper, we propose a new protocol called P^3 (Practical Packet Pipeline) that keeps its packet pipeline flowing despite the quality differences among channels. P^3 exploits sender and receiver diversities through synchronous transmissions (constructive interference), involving concurrent transmissions from multiple senders to multiple receivers at every stage of its packet pipeline. To optimize throughput further, P^3 uses *node grouping* to enable the source to transmit in every pipeline cycle, thus fully utilizing the transmission capacity of an underlying radio.

Our evaluation results on a 139-node testbed show that P^3 achieves an average goodput of 178.5 Kbps while goodput of the state-of-the-art high throughput protocol PIP (Packets In Pipeline) is only 31 Kbps. More interestingly, P^3 achieves a minimum goodput of about 149 Kbps, while PIP's goodput reduces to zero in 65% of the cases.

I. Introduction

Generating data in bulk is intrinsic to many sensor network applications such as monitoring of active volcanos, structural health, wildlife [32], [16], [3], [23], [7], and acoustic/image based sensing [24], [12], [1]. A model in which nodes sense and store the sensed data locally for later transfer in bulk is an attractive option for non-realtime sensor applications such as soil monitoring studies [27], [21], environmental monitoring [28], [30], and energy auditing [14]. This is because it has been shown that this model allows such applications to achieve ultra-low power consumption [20].

Transferring generated bulk data with high throughput can be important because: (1) High throughput reduces event miss rate [26]. Sensor nodes in applications such as monitoring of structural health and volcanos are required to suspend their event sampling during data transfer in order to avoid overwriting of the limited flash memory [32], [16]. (2) The time slots available for uploading of the collected bulk data can be short. For example, in railway-bridge monitoring [3], data is uploaded to passing trains. High throughput allows gathering and uploading of more data in the limited available time slots. (3) Ensuring a high throughput is an energy-saving technique as nodes can complete the data transfer faster and can go back to sleep for energy conservation [26].

The state-of-the-art approach to achieve high throughput in bulk data transfer in sensor networks is to exploit pipelining and channel diversity [22], [26]. The idea is to setup a *packet pipeline* by using different channels for different hops of a data collection route so that parallel transmissions can be accommodated without any self/intra-flow interference. There are two problems with this approach. First, it is vulnerable to the quality differences that exist among different channels. Our measurements demonstrate that the existence of even a single bad channel on one of the links can completely stall the packet pipeline resulting in zero throughput. Second, as the source transmits and the destination receives only in alternate cycles, the maximum achievable throughput is only up to half the link rate.

In this paper, we propose a Practical Packet Pipeline (P^3) that is robust to channel quality differences over multiple hops and allows the source to transmit and the destination to receive in every pipeline cycle so that the achievable throughput approaches link rate. The design of P^3 includes the use of packet pipelining, multiple channels, and synchronous transmissions. All or some of these concepts are also exploited in PIP [26] and Splash [5]. However, P^3 uses a number of novel techniques that are built on top of these concepts:

(1) We use synchronized multi-path transmissions to exploit both sender and receiver diversities so that the packet pipeline is robust to bad channels. P^3 is based on a key observation that packet receptions under such concurrent transmissions are not correlated. In P^3, we select a subset of relay nodes such that in every pipeline stage, except for the source and destination nodes, there are a minimum number of relays available for transmitting and receiving. As receptions are not correlated, with high probability, at least one of the receivers will correctly decode the packet. In this way, pipeline stalls (bubbles) are avoided as long as at least one node receives and forwards the packet at every stage. We call this *bubble-free pipelining*.

(2) We completely *fill up* the packet pipeline by enabling the source to transmit and the destination to receive in every pipeline cycle. To do so, we group nodes at each of the intermediate stages into 2 distinct subgroups, so that parallel transmission and reception can be accommodated in each cycle. While one subgroup receives, the other transmits. In this way, the full capacity of the underlying radio is utilized end-to-end.

(3) Finally, we exploit bubble-free pipelining even in the reverse direction of destination to source, for transmitting acknowledgements (NACKs). This is needed so that feedback on which packets to retransmit can be conveyed quickly and robustly by avoiding stalls caused particularly by channel asymmetry. With reliable feedback, retransmission can be performed

978-1-4799-3146-0/14/$31.00 ©2014 IEEE

only as needed without the need to resort to complex error correction and coding schemes.

We have implemented P^3 in Contiki-2.5 and evaluated our implementation on a 139-node testbed (Indriya) [4]. Our results show that P^3 achieves an end-to-end average goodput of 178.5 Kbps while PIP's average goodput is only 31 Kbps. This 5.7 times improvement is achieved despite of the fact that we reimplemented PIP and our reimplementation is 57% faster than its original implementation. More importantly, P^3 achieves a minimum goodput of about 149 Kbps, while PIP's goodput reduces to zero in 65% of the cases. Overall, the maximum observed end-to-end utilization of P^3 is 90.6%, with an average value of 80.9% of the effective link rate. This performance of P^3 is much higher than that of all the previous high throughput protocols [26], [22], [6], [15].

The rest of the paper is organized as follows. Background and related work are discussed in Section II. Section III presents a measurement study of performance differences that exist among different channels together with a study of reception correlation under synchronous transmissions on those channels. These studies serve as motivation for our work. We present P^3 and the details of its implementation in Section IV. Our evaluation results on the Indriya testbed are discussed in Section V. In Section VI, we provide a brief discussion on the extension of our work planned for the future. Finally, we conclude in Section VII.

II. Background and Related Work

In this section, we provide a discussion of the literature related to our work and background information on constructive interference that is exploited in our work.

Bulk Data Transfer. Bulk data in sensor networks is typically downloaded sequentially, from one node at a time. Only a single flow would be active at any given point of time so as to avoid inter-flow interference when multiple flows are active [15]. Sequential download also allows to achieve an ultra-low duty cycling [20], without increasing the overall completion time for downloading from an entire network [15]. These considerations drive existing high-throughput protocols [15], [22], [26], [6] to attempt to improve throughput over a single flow of bulk data transfer.

High-Throughput Protocols. While sequential download avoids inter-flow interference, self/intra-flow and external interferences still pose challenges for achieving a high throughput even on a single flow. A simple method to avoid intra-flow interference is to allow the source node to incorporate an inter-packet interval such that its previous transmission would be out of the interference range before the node attempts its next transmission. However, this method drastically reduces throughput as a long inter-packet gap is required in practice [16]. Flush [15] attempts to optimize this inter-packet interval by using an overhearing technique. But as typical interference range is more than the decoding (overhearing) range, Flush's approach is not effective in practice.

Osterlind *et al.* [22] are the first to propose to exploit channel diversity to completely avoid intra-flow interference in sensor networks. Their method involves assigning different channels for different hops of the active flow. As assigned channels are non-interfering with each other, nodes within a flow can make parallel transmissions without intra-flow interference. PIP [26] extends this idea to a full-fledged protocol which can achieve a high throughput (up to 63 Kbps) when the channel qualities over all hops are good. However, the existence of even a single bad channel on one of the hops can completely stall PIP's packet pipeline, reducing its throughput to zero.

Handling Channel Quality Differences. Channel hopping/cycling is a well-known technique that although reduces throughput, avoids the risk of packet pipelines being stalled completely. It involves changing the receiving channel of every hop in every pipeline cycle thus giving every hop chances to use good channels. However, unlike P^3, it is not a technique for ensuring a high throughput despite drastic channels-quality differences. For example, evaluations of PIP coupled with channel hopping [26] show that under intense interference, this combination can only achieve a goodput of below 15 Kbps.

While techniques like channel hopping does not ensure a high throughput, Duquennoy *et al.* proposed *Burst Forwarding* [6] so that nodes can at least conserve energy by switching to sleep mode (duty-cycling) during stalls. With duty-cycling, goodput of this protocol is lower than that of PIP. Finally, a naive method to handle channel quality differences is to use the maximum transmission power for data transfer over a route that is built using a lower transmission power or a poor channel. However, such an approach can induce self interference. Also, the underlying network may not be connected on lower power levels or poor channels.

Synchronous Transmissions (Constructive Interference). When multiple nodes transmit the same packet with strict synchronization, it can result in a physical phenomenon called constructive interference. A receiver node experiencing such interference can correctly decode the packet despite overlapping receptions. The key advantage of constructive interference is that it completely eliminates the need for contention resolution when multiple nodes have the same packet to transmit. Rahul *et al.* are the first to exploit this advantage in SourceSync [25] to improve the performance of WiFi networks. They improve download throughput in infrastructure networks and also enhance the performance of opportunistic routing [2] in ad hoc mesh networks.

Ferrari *et al.* are the first to demonstrate in Glossy [10] that strict synchronous transmissions lead to non-destructive interference even in wireless sensor networks. They showed that for the standard sensor radio (CC2420), if the maximum time displacement among concurrent transmissions of the same packet is less than 0.5 microsecond, with high probability, a receiver can correctly decode such transmissions. They exploited this fact to provide an efficient flooding and time-synchronization service. Splash [5] is another protocol that exploits constructive interference to rapidly disseminating large amounts of data to all nodes in a sensor network. Constructive interference is also used for collecting infrequent and periodic data in LWB [9] that allows to achieve a low duty-cycling ratio. Different from these uses, P^3 exploits constructive interference for achieving near optimal throughput in transferring bulk data between a source and a destination node.

Capture Effect Under Synchronous Transmissions. Landsiedel *et al.* recently demonstrated in Chaos [18] that the time required for all-to-all communication in sensor networks can be significantly reduced by exploiting capture effect under synchronous transmissions. The capture effect plays a key role in Chaos (it is mandatory) as all-to-all communication typically involves concurrent transmissions of *different* packets. However, its role under synchronous transmissions of the same packet is not as critical as in Chaos. It is demonstrated in Glossy [10] that reception reliability for identical packets is reasonably good without capture effect: 75% and 95% for packet sizes of 128 (maximum) and 8 bytes respectively. A strict synchronization among concurrent transmissions is the most critical requirement for identical packets as desynchronization even by a small value of 8 microseconds can degrade the reliability to below 15% [10]. Therefore, as P^3 involves only concurrent transmissions of the same packet, the requirement for synchronous transmissions (constructive interference) dominates over the need for capture effect.

Correlation among Packet Receptions. Recent works [33], [29] study reception correlation among different receivers of a packet transmitted by a single transmitter in sensor networks. These studies show that packet receptions are correlated on some channels whereas they are not on others. While Ting Zhu *et al.* exploit the observed correlation to improve the performance of network flooding [33], Srinivasan *et al.* attempt to quantify the correlation [29]. We study the correlation under constructive interference where a packet is simultaneously transmitted by multiple nodes to multiple receivers. Our measurement study demonstrates the fact that packet receptions under constructive interference are not correlated on all channels. Receptions are independent on sufficient number of channels so that P^3 can exploit node diversity through constructive interference to avoid pipeline stalls caused by channel quality differences.

III. Channel Quality and Reception Correlation Measurements

In this section, we present two measurement studies. First, we demonstrate that the performance of different channels on a link differs drastically. While it is known that the quality of a channel differs from that of another, we demonstrate that differences can be significant as a good link on one channel may not exist on another channel. Moreover, unlike previous works, we aim to provide extensive experimental evidence for the existence of the problem that we are tackling in this paper: channel quality differences can severely affect the performance of the state-of-the-art approach of exploiting channel diversity for bulk data transfer.

Second, we show that packet receptions under synchronous transmissions involving transmissions from multiple senders to multiple receivers are not correlated on all channels. While Marco Zimmerling *et al.* have shown in [34] that it is largely valid to assume packet receptions under synchronous transmissions to be independent, their experiments are limited to Channel 26. As P^3 is a multi-channel system and one of its *central* requirements is that the independence must be true on at least four different channels, we carryout experiments to demonstrate that this requirement can be satisfied in practice. This observation forms the crux of our proposed solution of P^3, allowing its pipeline to keep flowing.

A. Channel Quality Differences

IEEE 802.15.4 is the *de facto* physical layer standard for wireless sensor networks. This standard supports 16 non-overlapping channels defined in the 2.4 GHz band that is also shared by the WiFi devices. These 16 channels are usually referred by the numbers from 11 to 26, with each channel occupying a width of 2 MHz, and with an inter-channel space of 3 MHz. We evaluate performance of each of these 16 channels on links of the Indriya testbed [4], a 139-node sensor network deployed across three floors of a large building. We select links that are parts of the routes chosen by Contiki's collection tree routing protocol [17] as existing high-throughput protocols assume that their data routes are given by one such routing protocol. Moreover, we particularly choose high-quality inter-floor links which are critical in providing connectivity among nodes placed on different floors of the building.

Links are selected from two collection trees, one built on Channel 26 and the other on Channel 20. Both the channels are generally free from WiFi interference [19]. Channel 26 is used as the default channel in most of the sensor network deployments, and while network-wide quality of Channel 20 is generally not as good as Channel 26, it is one of those channels on which Indriya is consistently connected. On every link, we evaluate the performance of each of the 16 available channels. On each channel, we transmit 10,000 packets with an inter-packet interval of 10 milliseconds and use packet reception ratio (PRR) as an indicator of the channel quality.

Table I depicts the performance of all 16 channels (C) on 10 links (L) randomly chosen from the tree built on Channel 26. Each entry of the table corresponding to a link L and a channel C represents PRR observed over 10,000 packet transmissions made using channel C on link L. For example, the first entry (1, 11) whose value is 0.00 indicates that PRR observed for link numbered 1 on Channel 11 is zero. We can make two key observations. First, the quality of most of the other channels on all links are much lower than that of Channel 26. For example, on Link 1, not even one packet got through on any channel other than Channel 26 whose quality is almost perfect with a reception ratio of 0.99. Therefore, a good link on one channel may not even exist or it can be of bad quality on another channel (see last row that depicts percentage of links with the quality less than 0.1 ($\%_{<0.1}$)). As a result, when an existing protocol exploits a channel on a link that is chosen on some other channel, its packet pipeline can be stalled completely, resulting in zero throughput. The second observation is that no other channel is consistently as good as Channel 26. Thus making it difficult to find even a small set of channels that are good across most links.

Note that these observations were found true even on links which were chosen from a tree built on Channel 20. We found that Channel 20 is the only channel that is consistently good across those links (due to the lack of space, we are not presenting the corresponding data). Therefore, choosing routes on a somewhat poor channel where the network is consistently connected does not help existing protocols.

TABLE I: Quality differences among different channels on links of a tree built on Channel 26.

$L\backslash C$	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**
1	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99
2	0.23	0.00	0.00	0.00	0.00	0.00	0.37	0.51	0.72	0.03	0.38	0.26	0.00	0.00	0.01	1.00
3	0.82	0.01	0.00	0.21	0.53	0.67	0.01	0.37	0.26	0.00	0.14	0.30	0.00	0.36	0.97	0.99
4	0.00	0.00	0.28	0.50	0.49	0.00	0.05	0.26	0.69	0.00	0.00	0.02	0.00	0.00	1.00	1.00
5	0.00	0.00	0.00	0.00	0.00	0.51	0.47	0.49	0.59	0.00	0.03	0.10	0.00	0.06	0.94	0.98
6	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.00	0.08	0.91	0.50	0.48	0.40	0.36	0.77	0.92
7	0.72	0.01	0.00	0.02	0.10	0.00	0.00	0.03	0.07	0.01	0.01	0.01	0.01	0.00	0.08	1.00
8	0.09	0.38	0.36	0.70	0.97	0.55	0.50	0.47	0.16	0.52	0.26	0.18	0.18	0.29	0.99	1.00
9	0.11	0.23	0.41	0.97	0.99	0.48	0.47	0.45	0.03	0.09	0.01	0.30	0.46	0.91	0.65	1.00
10	0.69	0.73	0.73	0.23	0.11	0.23	0.15	0.36	0.95	0.83	0.15	0.00	0.40	0.69	1.00	0.96
Avg.	0.27	0.14	0.18	0.26	0.32	0.25	0.20	0.29	0.35	0.24	0.15	0.17	0.15	0.27	0.64	0.98
$\%_{<\mathbf{0.1}}$	50%	70%	60%	50%	40%	50%	50%	30%	40%	70%	50%	40%	60%	50%	30%	0%

TABLE II: Correlation coefficients observed on channels 15 and 20.

R	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	**1.0**	.02	0.0	.16	.08	.08	.01	0.0	.06	.12
2	.12	**1.0**	-.01	.13	.07	.05	.22	-.01	.06	.09
3	.14	.12	**1.0**	-.01	0.0	0.0	0.0	.07	.02	0.0
4	1.0	.12	.14	**1.0**	.25	.37	.13	.01	.20	.55
5	.16	.10	.13	.16	**1.0**	.10	.08	-.01	.26	.23
6	.18	.06	.11	.18	.08	**1.0**	.06	0.0	.09	.21
7	.24	.14	.17	.24	.17	.08	**1.0**	.01	.07	.09
8	.14	.08	.10	.14	.08	.05	.08	**1.0**	.01	.01
9	.30	.09	.08	.30	.21	.08	.19	.12	**1.0**	.18
10	.61	.07	.08	.61	.12	.11	.14	.09	.19	**1.0**

In summary, the-above observations demonstrate the fact that a common assumption of all channels behave similarly on a link chosen on some default channel does not hold in practice. The problem is that this assumption forms the crux of many existing protocols that are designed to achieve high throughput.

B. Correlation among Packet Receptions

Next we study the correlation among packet receptions under synchronous transmissions. Our experimental setup consists of 21 nodes that are installed on the same floor in an area of about 30m $\times$ 30m. These 21 nodes form a 2-hop setup in which one node acts as an initiator transmits a packet once every second to a set of 10 first-hop relay nodes, which in turn concurrently forward the received packet so that their transmissions interfere non-destructively at the remaining 10 second-hop receiver nodes (R), across which we study correlation. We repeat this experiment on four channels (15, 20, 25, and 26). Packets are transmitted for a duration of 5 hours on each channel. We particularly choose these channels because they are generally non-overlapping with commonly used WiFi channels [19].

In order to quantify correlation, we compute Pearson's correlation coefficient at a packet-level granularity. For illustration, Table II depicts the coefficient values observed for Channels 15 and 20. Note that as a coefficient matrix corresponding to a channel is symmetric, we represent data corresponding to two channels in a single table (matrix). The lower half of the table (below the diagonal) corresponds to Channel 15 and the upper half corresponds to Channel 20.

It can be observed that the coefficient values are generally small for every channel with at least 80% of the values being less than 0.2. This indicates that when packet reception is unsuccessful on one of the receivers, some other receivers may be able to receive the packet successfully. Similar coefficient values were observed for the other two channels numbered 25 and 26. However, note that this is not true across all 16 available channels. For example, since Channel 22 typically overlaps with a commonly used WiFi channel, it experiences relatively strong correlation. On the other hand, since channels 15, 20, 25 and 26 are generally not affected by the WiFi, the correlation is low.

As long as there exists at least four such channels with low packet reception correlation under synchronous transmissions, P^3 can successfully exploit node diversity through synchronized multi-path transmissions to keep the pipeline flowing without stall.

IV. $\mathbf{P^3}$: A High Throughput Protocol

The state-of-the-art protocol to achieve high throughput in sensor networks is PIP [26]. Given a collection route chosen on some default channel (see Fig. 1(a)), the key idea of PIP, which is adopted from [22], is to setup a packet pipeline by assigning different channels to different hops of the route. However, the problem with this approach is that it often leads to a scenario depicted in Fig. 1(b), where no packet gets through from node 1 to 2 on the assigned Channel C2. This completely stalls PIP's packet pipeline to result in zero throughput at the destination.

In order to tackle this problem, we propose P^3, a practical high-throughput protocol that keeps its packet pipeline flowing despite drastic channels-quality differences.

P^3 is mainly a combination of three key innovations:

1) *Bubble-free pipelining* exploits both sender and receiver diversities through synchronous transmissions to avoid bubbles (stalls) over a multi-channel packet pipeline.
2) *Pipeline filling* allows to create a packet pipeline in which the source transmits a packet in every cycle, similar to a CPU-instruction pipeline where an instruction is fetched and fed in each cycle. On the other hand, existing packet pipelines allow source to transmit a packet only in alternate cycles.
3) *Bi-directional pipelining* supports pipelined transmissions of acknowledgement messages (NACKs) exploiting node diversity in the reverse direction from destination to source so that the stalls due to link asymmetry are effectively avoided.

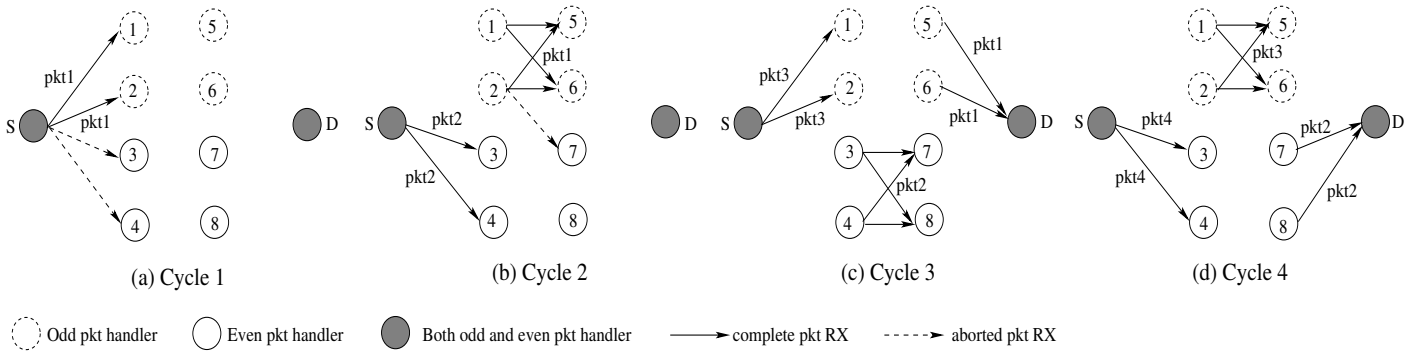
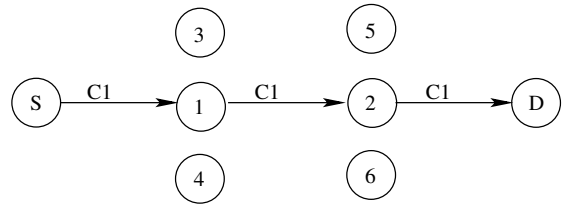
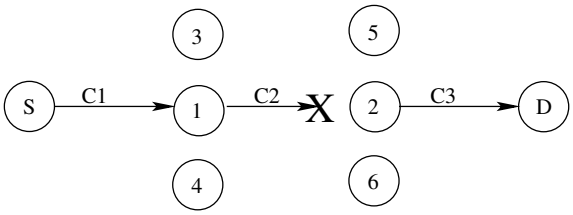


Fig. 3: P^3's packet pipeline achieving the maximum possible end-to-end throughput.



(a) Default data route chosen on the default channel (C1).



(b) Channel Assignment by PIP on a given route.

Fig. 1: Problem with the state-of-the-art approach to achieve high throughput.

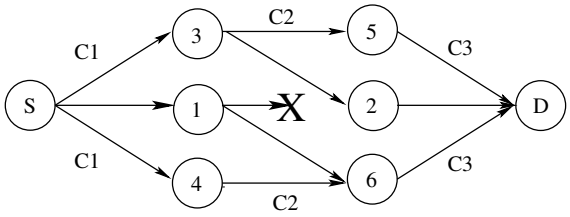


Fig. 2: Proposed solution of exploiting node diversity through synchronous transmissions.

In the rest of this section, we discuss various components of P^3 and some of its implementation details.

A. Bubble-Free Pipelining

P^3's core technique is to exploit node diversity through synchronous transmissions, so that pipeline bubbles can be avoided despite drastic channels-quality differences. This is depicted in Fig. 2. At every intermediate stage, there are multiple nodes, all of which forward a received packet, so that at least one of such senders would successfully communicate to at least one of the multiple receivers available at the next hop. This increases the chance that a transmission between two hops is successful. Such transmissions using multiple senders and receivers are said to exploit sender and receiver diversities respectively [25], [2]. For illustration, consider Fig. 2 in which although there is no communication link on the assigned Channel C2 from node 1 to 2, other links such as 4-to-6, 3-to-2, 3-to-5, and 1-to-6 are of good quality on the same channel. All that is needed is that at least one transmission succeeds so that the pipeline can continue without creating a bubble.

A key question while exploiting node diversity is that when and which of the multiple receivers of a packet should forward the received packet? Conventional approaches used an agreement protocol to select such a forwarder [2]. However, such an agreement protocol is typically expensive, trading both throughput and energy. Instead, we exploit synchronous transmissions as in [25] to completely eliminate the need for such an agreement protocol. On receiving a packet, all receivers forward the packet immediately and synchronously so that their transmissions interfere non-destructively at the next-hop receivers. As packet receptions under synchronous transmissions are not correlated (see Section III), with high probability, at least one of the receivers would correctly decode the packet.

B. Pipeline Filling

Existing protocols can at most utilize 50% of the capacity of an underlying radio as their source node can only transmit in alternate pipeline cycles where the first-hop node is listening, not transmitting. On the other hand, the source node in P^3 transmits every cycle, utilizing the radio to its full capacity.

Let us describe how P^3 achieves this using an example network of 10 nodes depicted in Fig. 3. To begin, source (S) transmits the first packet (pkt1) in the first pipeline cycle (see Fig. 3(a)). Although this packet is heard by all the first-hop nodes, only nodes 1 and 2 choose to completely receive the incoming packet while others abort the reception midway. In the second cycle (see Fig. 3(b)), the next transmission (pkt2) immediately follows from the source, and this time nodes labelled as 3 and 4 receive pkt2 while nodes 1 and 2 are busy in forwarding pkt1 synchronously on another channel. So the first-hop stage is both transmitting and receiving in the same cycle. Moreover, source does not waste the second cycle as in existing protocols. By grouping nodes at each intermediate stage into two subgroups (one receiving and transmitting packets which source generates in odd-numbered

pipeline cycles and the other handling packets generated in even-numbered cycles), once the pipeline is filled, P^3 keeps every stage busy in every cycle: the source transmits, an intermediate stage both receives and transmits, destination receives (see Figs. 3(c) and (d)).

C. Selection of Data Routes

In this subsection, we explain the procedure that P^3 uses for selecting its data routes between a given source and a destination node.

1) **Network Flooding is Inefficient***:* Network flooding over synchronous transmissions offers the maximum node diversity at every hop. However, it has two key drawbacks: (1) it requires an entire network to be awake for transferring data between only two nodes, which is highly inefficient in terms of energy consumption; (2) it is well-known that reception reliability of synchronous transmissions suffers from *scalability problem*, both in terms of number of concurrent transmitters and packet size [31], [5]. Most of the previous protocols [10], [9], [5] exploiting flooding over synchronous transmissions rely on multiple retransmissions of *every* packet, for higher reliability. However, such a retransmission policy is not an option for P^3 as it drastically reduces throughput by a factor of the employed number of retransmissions.

2) **Finding Multiple Node-Distinct Paths***:* Instead of using the maximum diversity offered by flooding (all network nodes), P^3 exploits node diversity available in a much smaller sub-group of nodes that form a set of node-distinct paths, of a certain hop-length (H), between the given source and destination nodes. This is illustrated in Fig. 4. Fig. 4(a) shows a sample network topology used to illustrate the route selection process. There are two special nodes, the source (S) and destination (D). For all other nodes, the number indicated within each node is the hop-length from the source node.

Provided with this global topology at the gateway node, it finds distinct paths as follow. First, it identifies the shortest path from node S to node D (obtain using a standard shortest path algorithm), which is 3 hops in this case. One can now ask for node-distinct paths with 3 or more hops. In our current implementation, for a given value of H, the gateway uses a depth-first-search algorithm to find the largest set of available node-distinct paths of the given length H. For $H = 3$, two such paths can be found for the sample network as shown in Fig. 4(b). All other nodes that do not lie on these selected paths will not participate in the data transfer, thus significantly reducing the overhead.

A key question is how many such node-distinct paths can be found (and is needed) between any given two nodes so that the selected set of nodes offer sufficient node diversity? Note that while a logical value for H is the hop-length of the shortest path between the source and destination nodes, we can always choose to use a different H so that the number of distinct paths (diversity) available changes. Also note that hop-length has minimal effect on the performance of P^3 because of the use of pipelining. We select H such that there should be *at least* N distinct paths of length H. Typically, there will be several sets of distinct paths between the given source and destination, with each set corresponding to a different hop-length. Hence, there is a high chance of finding at least one set among them having at least N distinct paths.

A fundamental assumption we made in the design of P^3 is that the network deployment is dense. In the testbed we used to evaluate P^3 (Indriya), the average number of neighbors per node (density) is about 20. Node density is even higher for other popular testbed deployments such as Twist [13] and Kansei [8]. Based on our experimental observations, we choose a value of six for N. Therefore, P^3 exploits *at least* six (N) distinct paths for creating its packet pipeline (see Table IV in Section V for number of distinct paths ($M \geq N$) forming our experimental routes).

Nevertheless, it is always possible that the network is not sufficiently dense and the number of paths with distinct nodes is small. In such cases, P^3's performance degrades and in the worst case, the performance is no worse than the performance of our baseline protocol, PIP, which uses only a single path.

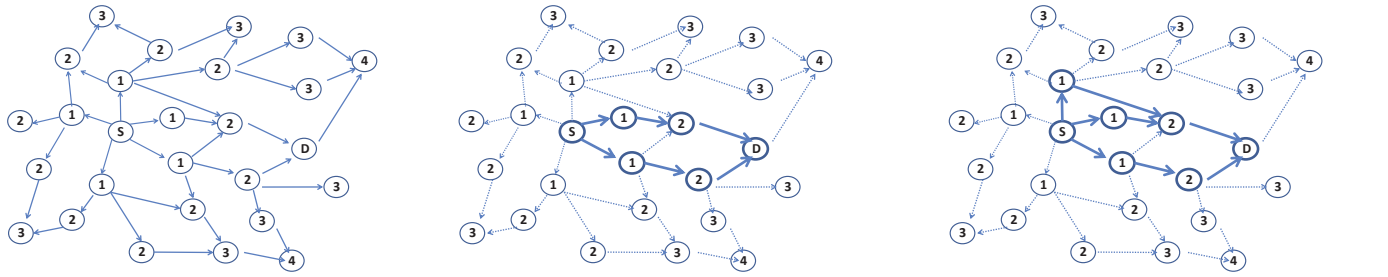
3) **Selection of Even and Odd Forwarding Nodes***:* If there are M ($M \geq N$) distinct paths available between a given source and destination, a subset of $M/2$ paths receive and forward odd-numbered packets (i.e., packets which are generated in the odd-numbered pipeline cycles at the source) and the remaining paths handle even-numbered packets. We build these subsets by choosing paths randomly among the available M paths. In addition, our current implementation of P^3 also supports increasing node diversity by adding extra nodes. For example, one additional node can be added as shown in Fig. 4(c) to the first hop in addition to the set of nodes shown in Fig. 4(b). Its addition increases receiver and sender diversities at the first and second hops respectively.

D. Overhead of Route Selection

All bulk data collection protocols need a routing protocol for selecting routes over which bulk data is downloaded. There are two choices for such a routing protocol: (1) a conventional distributed protocol like CTP [11], as assumed by systems such as Flush [15] and PIP [26]; (2) a centralized approach as proposed in Koala [20], which involves gateway node collecting qualities of all the links of every node in the network and using the collected information for computing data routes. In other words, gateway node needs have the knowledge of the global topology.

For P^3, we adopt the centralized approach because (as demonstrated in Koala), while this approach incurs an overhead when collecting network topological data, it still can be more efficient than a distributed protocol that incurs the overhead of persistently maintaining routing state, which is not needed for infrequent transfer of bulk data. We will discuss in more detail the two main sources of overhead (1) collection of link quality statistics to the central gateway and (2) dissemination of routing information to the network nodes.

1) **Link Quality Collection***:* Collection of link qualities involves two phases: (a) measuring the link qualities and (b) collecting the measured data from every network node to the gateway.



(a) A sample network. (b) Two distinct paths of hop-length 3. (c) Two distinct paths with added diversity.

Fig. 4: P^3 Data Route Selection.

Link Quality Measurement. The overhead incurred in link quality measurement is common to any bulk data transfer protocol. The additional requirement for P^3 is that this link quality measurement should be done quickly so that the measurements remains useful for P^3 during the data transfer phase.

The proposed approach uses a Glossy-like [10] efficient synchronization protocol and it involves every network node broadcasts a certain number of beacon packets to its neighbors so that they can compute qualities of the incoming links. A key requirement for this method is the need for orchestrating broadcasts from one node at a time, for which we propose to use Glossy that limits orchestration overhead to only a few milliseconds per node. Based on the capability to achieve efficient synchronization, we can write down a simple equation (shown below) that represents the total time required for measuring link qualities from a network of size n nodes. In the equation, T_o is the time required to disseminate a control packet to a network node using a protocol like Glossy, indicating the node to start broadcasting. IPI represents inter-packet interval used for transmitting beacons, and N_b is the number of such beacons transmitted.

$$T = (T_o + (IPI \times N_b)) \times n \tag{1}$$

Using the above equation and making an assumption that T_o is less than 10 milliseconds [10], measuring link qualities of a 100-node network with a diameter of up to 10 hops would take only about 101 seconds, assuming $IPI = 100$ milliseconds and $N_b = 10$.

Collection of Measured Link Qualities. We propose to make use of an efficient collection protocol such as Low-Power Wireless Bus (LWB) [9]. It has been shown in [9] that LWB can complete the collection of 15-byte packets from a network of 260 nodes within only about 5 seconds. Hence, it is reasonable to assume that for network sizes of up to 260 nodes, an overhead of no more than 10 seconds is needed to collect link statistics from all nodes with each node transmitting a bigger 30-byte packet accommodating more link entries.

2) **Dissemination of Routing Information and Channel Assignment***:* Once gateway node computes distinct paths, we exploit synchronous transmissions for disseminating the computed data to the network so that the paths are installed (network nodes which are part of the computed paths are informed to get ready for the data download that follows).

The disseminated routing data contains node identity, hop-length, and reception channel for every node of the set of node-distinct paths. Nodes of the same hop-length use the same reception channel. Nevertheless, different channels are used for different hops so that packet pipelining can be supported. We exploit only those four ZigBee channels (15, 20, 25, and 26) which are typically free from WiFi interference [19]. This is because such a ZigBee channel exhibits no positive correlation among its packet receptions under synchronous transmissions (see Section III), which is a key requirement for P^3. Importantly, due to node diversity available at every stage of the P^3's pipeline, only four such channels are sufficient to cope with any self interference on routes longer than 4 hops.

Dissemination involves Glossy-like [10] flooding, which completes within a few tens of milliseconds, thus minimally affecting the overall download throughput.

3) **Impact of Variations in Topology***:* A key issue is whether the collected topology undergo significant changes in the period between measuring-and-collecting link qualities and the completion of bulk data transfer from every node in the network.

It is easy to infer from equation 1 and assuming LWB for collection that it takes at most 111 seconds for measuring and collecting link statistics from a 100-node network (101 seconds for link quality measurement (beacon $IPI = 100\ milliseconds, N_b = 10$, and $T_o = 10\ milliseconds$) and 10 seconds for collection using LWB). Moreover, based on P^3's average throughput of about 178.5 Kbps (as observed in our experiments that takes dissemination time of routing information into account), P^3 requires about 287 seconds for downloading a 64 KBytes object from every node in a network of 100 nodes. Therefore, total data download completion time is about 400 seconds. It is reasonable to assume that the global topology does not drastically change over these relatively short durations.

E. Scalability at the Last Stage

As there are multiple senders and receivers between any two intermediate stages of P^3's pipeline, with high probability, at least one of the receivers will correctly decode the packet.

However, the last stage (destination node), lacking any receiver diversity, often experiences poor reception reliability due to too many concurrent senders causing the scalability problem [31]. To tackle this, we use a simple technique in which only those nodes on the penultimate stage with a good reception reliability of at least a predefined threshold value would participate in forwarding the received packets. This typically limits the number of concurrent transmissions overlapping at the destination.

F. Bi-Directional Pipelining

Packet losses are common in wireless networks and missing data has to be recovered. Rateless/XOR coding is not suitable as these techniques involve decoding overhead that reduces throughput considerably. As there exists only one destination, the source can have accurate information of missing packets. We use a simple retransmission policy involving transmission of NACKs indicating missing packets in the form of a bit vector. The key is that even NACKs are transmitted over a bubble-free pipeline exploiting node diversity, but in the reverse direction from destination to source. This is particularly useful as existing pipelines lacking such diversity often fail to communicate NACK-like control messages due to the effects of channel asymmetry on their links.

Source starts the retransmission procedure by transmitting a control message that immediately follows the transmission of the last data packet. It is conveyed over the same bubble-free pipeline used for data. This control message serves two purposes of informing the destination to transmit a NACK and notifies intermediate nodes to change their sending channels so that the NACK can be transmitted over pipelining. As a response to the reception of a NACK, intermediate nodes forward it and switch back to their original sending channel, and the source starts retransmitting missing packets. The entire process is repeated until the destination receives all packets.

G. Implementation of $\mathbf{P^3}$

P^3 is implemented in Contiki OS based on code from Glossy [10] and Splash [5]. In particular, support for synchronous transmissions is derived from Glossy, and the capability to exploit multiple channels while accommodating synchronous transmissions is derived from Splash's code. In this section, we provide an overview of the implementation aspects that are specific to P^3.

Packet Transfer from MCU to Radio[1]. As the source node in existing approaches alternates between *transmission busy* and idle cycles, the time taken for loading a packet from MCU to radio memory has no impact on the end-to-end throughput. This is because a packet can be loaded in an idle cycle for its transmission in the proceeding busy cycle. On the other hand, P^3 allows the source to transmit a packet in every cycle. Therefore, waiting until a packet is entirely loaded into radio memory before starting its transmission considerably degrades its throughput. Note that packet loading takes considerable amount of time but typically less than the duration that radio requires for packet transmission.

[1]Note that this issue is different from the bottleneck issue of SPI (Serial Peripheral Interface) discussed in PIP [26].

We tackle this issue by parallelizing loading of a packet with its own transmission. We start loading a packet immediately after sending a command strobe to the radio for its calibration, and radio starts transmitting the packet as soon as it is calibrated while packet's loading is still in progress. As loading is faster than transmission, buffer underflow is not an issue [10]. The same technique is exploited in Glossy [10] but for a different purpose of ensuring synchronization for synchronous transmissions.

Avoiding Explicit Synchronization. Existing high throughput protocols require their pipeline nodes to be tightly synchronized to one another so that their channel switching can be coordinated. Such synchronization is ensured and *maintained* by explicit regular transmissions of timestamps by the source to all the other nodes. On the other hand, P^3 supports the required synchronization at no additional cost and its explicit maintenance is also not required.

This is because P^3 is built on the code for Glossy and Splash [10], [5], which allows nodes to ensure a highly deterministic and constant delay in a sequence of operations of receiving a packet, switching to the transmit channel, forwarding of the received packet, and finally switching back to the receiving channel. When such determinism is coupled with strictly regular transmissions from the source and packets of equal length, by the time a pipeline stage starts its next transmission, its next stage receiver nodes would have completed forwarding of the previous packet and switched back to their receiving channel.

Lack of OS Services. As P^3 exploits synchronous transmissions for both data and control packet transmissions, its code does not use any service from its OS (Contiki) as such services incur variable delays making it impossible to support synchronous transmissions. P^3 is implemented entirely as part of an interrupt handler with all the other interrupts being disabled during P^3's operation. Lack of OS services such as abstractions that expose any number of timers made implementing P^3 a difficult task. Particularly, all the required timeouts for control operations had to be implemented using only two timers out of only six available timers in total, while the remaining four were used for mainstream data operations.

Flash Access. As P^3 achieves almost maximum goodput, packets are always being received at the destination node. Thus it is challenging for this node to find sufficient amount of time so that received packets can be written into the external flash or transmitted over the serial port. One option is to overlap reception of a byte by radio with writing of the immediate previously read byte into flash. As flash access is faster than the radio reception, MCU can become free in time to read the next radio byte. Another simpler option is to have a set of two sensor nodes as the base station (destination), in which one node receives packets in the odd-numbered cycles and the other in the even-numbered cycles. This gives each node sufficient time duration to write a received packet into the flash or transmit it onto the serial port while the other node is busy in receiving a packet over the radio.

However, for simplicity, in our current implementation of P^3, we assume that the base station has enough RAM memory to hold all the bulk data from all the network nodes.

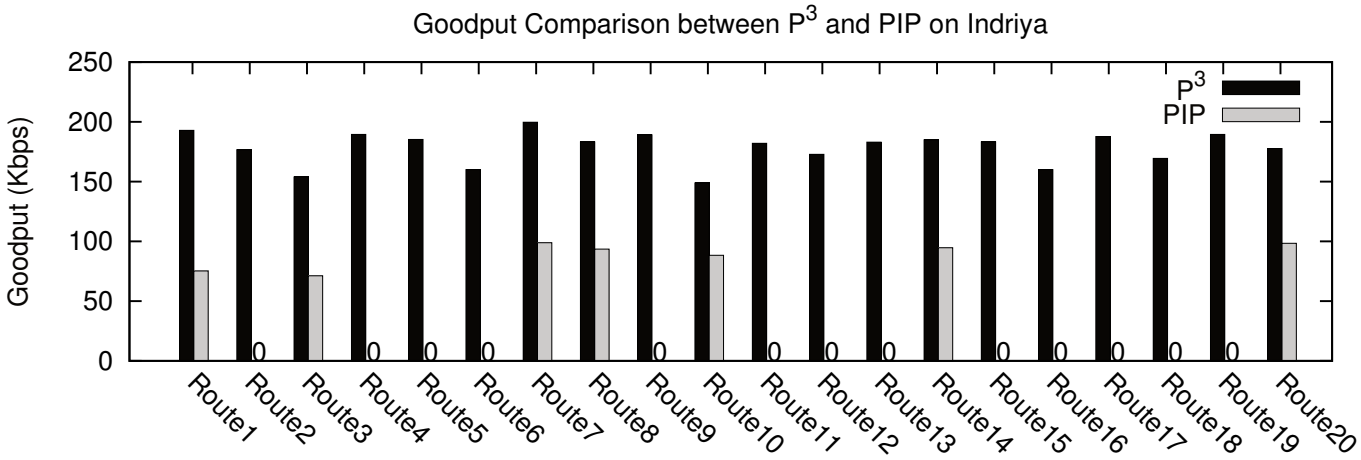


Fig. 5: Goodput comparison between P^3 and PIP.

V. Performance Evaluation

In this section, we present the results of our experiments carried out on Indriya [4], a 139-node facility available at the National University of Singapore. Indriya uses TelosB nodes built with CC2420 radio chips and is deployed over three floors of a building.

We compare P^3 against PIP [26]. We reimplemented PIP in Contiki based on Glossy code [10] and it is much faster than all the previous implementations of PIP. While the original implementation can at most achieve a goodput of 63 Kbps [26], [6], our reimplementation can achieve the maximum goodput of about 99 Kbps. This improvement is due to the fact that our code is based on Glossy's code that ensures highly deterministic and minimum delays in processing packets. Note that PIP was also reimplemented by Duquennoy *et al.* in Burst Forwarding [6] and their implementation can at most reach a goodput of 73 Kbps. Similar to the reimplementation by Duquennoy *et al.* [6], our code for PIP is also a basic version that does not support supplementary techniques such as channel hopping. These missing techniques do not address the effects of drastic quality differences that exist among different channels, which is the main goal of P^3. For simplicity, we use our testbed's USB-backend for collecting all the required data representing global topology of the underlying network. Nevertheless, we provide an analysis of the overhead incurred when a collection protocol is used for collecting such data.

For all our experiments, we use the maximum transmission power of 0 dBm and distinct routes on which pipelined transmissions are carried out are chosen on Channel 26 which is the most commonly used default channel. For both P^3 and PIP, we use the same set of channels to support pipelining. For P^3, we choose multiple routes as explained in Section IV. For fair comparison, we randomly choose one among these routes for PIP and retain the same assignment of channels as used in P^3. We make sure that these routes are built using high-quality links (with ETX 1). For packet size, we use the maximum possible value of 128 bytes for both the protocols (as supported by the CC2420 radio), with each packet containing a data payload of 118 bytes. For every experimental run, we transmit 500 such maximum-sized packets for a total bulk data size of 59 KBytes. Moreover, content of every packet is randomly generated. For metric, we measure goodput, the number of application-data bits successfully received at the destination per unit time. It is computed by measuring time duration that starts with transmission of the first wakeup packet by the root (base station/destination) that contains routing information and channels to be used for pipelined transmissions. The duration ends when the root successfully receives the last data packet.

Summary of Testbed Results. Fig. 5 compares goodput achieved by P^3 and PIP on 20 routes whose source and destination nodes are located on different floors of Indriya in most cases. Hop-count for these routes ranges from 2 to 6 hops with an average value of 4.13 hops per route. Note that typical maximum hop-count for common routing protocols like CTP [11] on Indriya is 5 or 6 hops at 0 dBm. We can make a few observations from the figure. First, PIP's goodput often reduces to zero as observed on 13 out of 20 cases (e.g., Route2). This is due to the lack of a communication link on at least one of the channels assigned for one of the PIP's pipeline stages (hops), which in turn completely stalls its pipeline resulting in a goodput of zero. On the other hand, as P^3 exploits node diversity, it is able to achieve an average goodput of 179.1 Kbps in cases where PIP's goodput is zero.

The second observation is that link asymmetry is a major problem for PIP as it can also completely stall its pipeline. For example, on Route9, while an assigned channel over a certain hop experiences good quality in the forward direction from source to destination, there is no communication link in the opposite direction for control packets such as NACK messages that requests retransmission of missing packets. Lack of control messages prevents the source from transmitting any missing data reducing goodput to zero. P^3 handles asymmetry effectively as it exploits node diversity in the opposite direction from destination to source, thus resulting in a goodput of 189.1 Kbps on the same route (Route9).

Overall, the minimum and maximum goodput values observed for P^3 are 148.8 Kbps (Route10) and 199.7 Kbps (Route 7) respectively. On average, P^3 achieves a goodput of

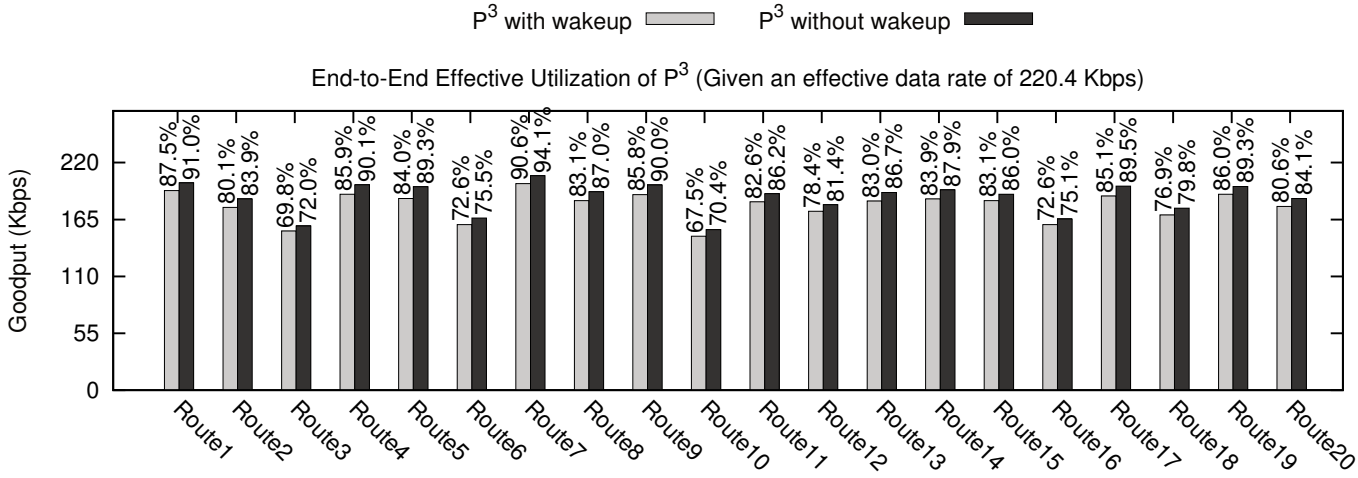


Fig. 6: End-to-End Effective Utilization of P^3.

TABLE III: Analysis of Transmission Rounds in P^3.

Route No.	N_{TXR}	$R1$ (%)	P_{RTX}(%)	G (Kbps)
1	1	100.0	0.00	199.74
2	3	89.60	9.74	165.32
3	2	99.40	0.59	192.18
4	4	80.20	19.22	147.13
5	2	98.40	1.57	188.42
6	4	76.60	21.50	142.94
7	2	99.40	0.59	189.40
8	3	90.20	9.58	166.96
9	1	100.0	0.00	197.40
10	3	87.40	12.12	162.98
Avg.	2.5	92.12	7.49	175.25

178.5 Kbps whereas PIP's average goodput is a much lower value of only 31 Kbps.

Effective Utilization. In order to measure utilization, we need to know the *effective physical data rate* supported by an underlying radio. It is defined as the maximum speed at which a radio can transmit application bits. From our measurements, it is 220.4 Kbps for the *de facto* standard CC2420 radio. Given this rate, we define *effective utilization* as the percentage of this capacity that P^3 utilizes in exclusively serving application data end-to-end.

Fig. 6 depicts effective utilization of P^3 for two of its cases: (1) includes the time taken for wakeup that control packet dissemination; (2) does not consider overhead, only complete end-to-end transmission of bulk data. As shown, utilization can reach up to 90.6% as observed on Route 7. The lowest utilization of 67.5% can be seen on Route10. Nevertheless, the goodput on Route10 is still significantly high at 148.8 Kbps. On average, 80.9% of the effective data rate is utilized by P^3 to serve its application data. The remaining capacity of 19.1% is mainly taken by P^3's 7-bytes application header, wakeup, and retransmissions.

Transmission Rounds. Table III depicts the total number of transmission rounds (N_{TXR}) that P^3 needs to complete the data transfer, for another set of 10 random routes. The table also depicts $R1$, the reliability observed at the destination after the first round that involves transmission of an entire bulk object, P_{RTX} is the percentage of retransmissions in

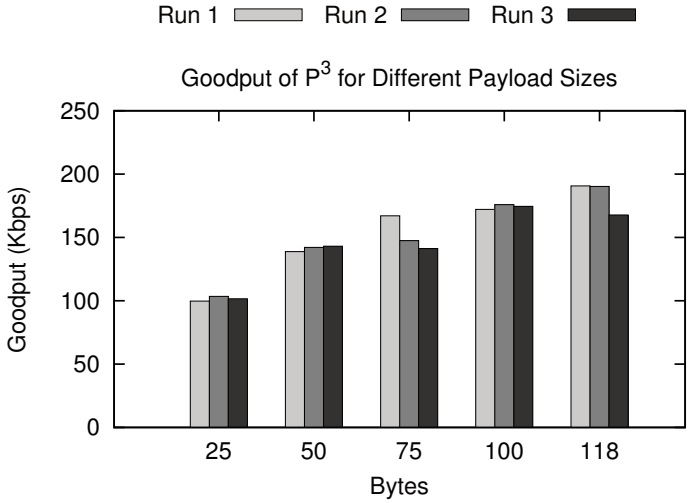


Fig. 7: Goodput of P^3 for different packet payload sizes.

the total number of data transmission attempts made over all rounds, and G is the goodput. As shown, P^3 takes up to four transmission rounds as can be observed on the 4^{th} and 6^{th} routes. For $R1$, its average value is a good reliability of 92.12%, showing that P^3 effectively exploits the benefits of node diversity. The lowest reliability of 76.6% can be observed on the 6^{th} route, where retransmissions constitute about 21.5% (P_{RTX}) of the total transmission.

Finally, the goodput (G) ranges from 142.9 to 199.7 Kbps, with an average value of 175.2 Kbps. P^3 is able to achieve such high goodput as pipeline stalls in all of its multiple transmission rounds are avoided by its bubble-free pipelining, exploited in *both the directions* of source to destination and the reverse.

Effect of Packet Size. There exists a well-known tradeoff in wireless communications that while increasing packet size can reduce control overhead such as headers and preambles, it increases the probability that a packet gets corrupted. This is particularly true under synchronous transmissions [5]. So we execute P^3 configured with 5 different payload sizes. Fig. 7

TABLE IV: Number of active nodes and node-distinct paths exploited in PIP and P^3.

Route No.	H	NA_{pip}	NA_{P^3}	M (P^3)
1	3	4	14	6
2	5	6	26	6
3	2	3	10	8
4	5	6	30	7
5	5	6	30	7
6	4	5	23	7
7	2	3	8	6
8	4	5	29	9
9	3	4	14	6
10	4	5	20	6
11	6	7	37	7
12	6	7	32	6
13	5	6	26	6
14	5	6	30	7
15	5	6	30	7
16	5	6	34	8
17	4	5	32	10
18	5	6	26	6
19	5	6	30	7
20	4	5	26	8

plots P^3's goodput for the considered sizes, with three runs in each of the cases. As we can see, goodput is maximum for the maximum-sized payload of 118 bytes as gain that is rendered by a larger packet outweighs the loss that it incurs due to corruption. Therefore, we use the maximum value as the default payload size in P^3.

Number of Active Nodes and Energy Consumption. An obvious observation is that P^3's multi-path pipeline requires more nodes to participate in data transfer compared to what is required to build PIP's single-path pipeline. Table IV compares the number of active nodes (NA_{pip} and NA_{P^3}) for PIP and P^3 on the 20 considered routes. The table also includes other statistics such as hop-length (H) and the number of node-distinct paths exploited in P^3 routes (M). We can see that P^3 needs on the average 4.7 times more active nodes than PIP. However, at the same time, P^3's average goodput is 5.7 times more than PIP (see Fig. 5).

As energy consumption is directly proportional to throughput, P^3 consumes lesser energy than PIP. For example, power consumption for a PIP's route of 6 nodes is about 5.59 Watts whereas P^3 although exploits a more number of about 28 nodes on the same route, it consumes a lesser energy of 4.56 Watts. These power consumption values are calculated based on the measured average number of active nodes and throughput, a 64-KBytes data object, and the power consumption specification of the underlying CC2420 radio. Therefore, P^3 offers a much higher goodput at a lower energy consumption than PIP for downloading the same amount of data. For applications that require shorter completion time, P^3's much higher throughput is clearly more desirable.

Routing Overhead. As the overhead of link quality measurement is common to any bulk data transfer protocol, similar to recent protocols of PIP [26] and Bursty Forwarding [6], we do not include this overhead. Instead, we only consider the link statistics collection time that is specific to P^3. Assuming that the LWB collection protocol [9] is used, it would require only up to 10s for collecting the measured information from an entire network of 260 nodes [9]. Therefore, given the fact that P^3 achieves an average goodput of about 178.5 Kbps, it requires about 746 seconds to download a 64-KBytes object from every node of the same 260-node network. Therefore, a 10 seconds overhead required for collecting link statistics adds only about 1.32% to the total data download time.

VI. Future Work

Ensuring high packet reception reliability under varying topological and channel conditions is a challenge for any wireless system. In addition, synchronous transmissions are more prone to corruption as additional factors such as number of concurrent transmitters can also affect the reliability. While these factors affect most of the previous protocols exploiting synchronous transmissions [10], [9], [5], they also affect P^3. We are currently working on an adaptive method to learn about the best channel and nodes at a hop which should be exploited for concurrent transmissions (forwarding) of a received packet so that the average reception reliability at the next hop is maximized. This future work will be useful in general to protocols that exploit synchronous transmissions and which rely on a fixed number of packet retransmissions for high reliability.

VII. Conclusion

Due to drastic quality differences that exist among different channels, performance of the state-of-the-art approach of exploiting channel diversity to create a high-throughput packet pipeline is often poor in practice. In order to tackle this problem, we have proposed P^3, a practical packet pipelining protocol that keeps its packet pipeline flowing despite channels-quality differences. Our key technique is to exploit both receiver and sender diversities through constructive interference. Moreover, unlike existing approaches whose maximum achievable throughput is only half the link rate, P^3 can closely reach the link rate. Our evaluation results on a 139-node testbed show that P^3 achieves an average goodput of 178.5 Kbps while PIP's average goodput is only 31 Kbps. More interestingly, P^3 achieves a minimum goodput of about 149 Kbps, while PIP's goodput reduces to zero in 65% of the cases.

Acknowledgements

We would like to thank the anonymous reviewers and our shepherd, Bhaskar Krishnamachari for their comments and suggestions.

References

[1] P. Bergamo, S. Asgari, H. Wang, D. Maniezzo, L. Yip, R. E. Hudson, K. Yao, and D. Estrin. Collaborative Sensor Networking Towards Real-Time Acoustical Beamforming in Free-Space and Limited Reverberance. *Mobile Computing, IEEE Transactions on*, 2004.

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

[3] K. Chebrolu, B. Raman, N. Mishra, P. K. Valiveti, and R. Kumar. Brimon: A Sensor Network System for Railway Bridge Monitoring. In *Proceedings of MobiSys*, 2008.

[4] M. Doddavenkatappa, M. C. Chan, and A. Ananda. Indriya: A Low-Cost, 3D Wireless Sensor Network Testbed. In *Proceedings of TRIDENTCOM*, 2011.

[5] M. Doddavenkatappa, M. C. Chan, and B. Leong. Splash: Fast Data Dissemination with Constructive Interference in Wireless Sensor Networks. In *Proceedings of NSDI*, 2013.

[6] S. Duquennoy, F. Österlind, and A. Dunkels. Lossy Links, Low Power, High Throughput. In *Proceedings of SenSys*, 2011.

[7] V. Dyo, S. A. Ellwood, D. W. Macdonald, A. Markham, C. Mascolo, B. Pásztor, S. Scellato, N. Trigoni, R. Wohlers, and K. Yousef. Evolution and Sustainability of a Wildlife Monitoring Sensor Network. In *Proceedings of SenSys*, 2010.

[8] E. E, A. A, R. R, and N. M. Kansei: A Testbed for Sensing at Scale. In *Proceedings of the IPSN*, 2006.

[9] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Low-Power Wireless Bus. In *Proceedings of SenSys*, 2012.

[10] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient Network Flooding and Time Synchronization with Glossy. In *Proceedings of the IPSN*, 2011.

[11] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection Tree Protocol. In *Proceedings of SenSys*, 2009.

[12] B. Greenstein, A. Pesterev, C. Mar, E. Kohler, J. Judy, S. Farshchi, and D. Estrin. Collecting High-Rate Data over Low-Rate Sensor Network Radios. Technical report, 2005.

[13] V. Handziski, A. Kopke, A. Willig, and A. Wolisz. TWIST: A Scalable and Reconfigurable Testbed for Wireless Indoor Experiments with Sensor Network. In *Proceedings of REALMAN*, 2006.

[14] X. Jiang, M. Van Ly, J. Taneja, P. Dutta, and D. Culler. Experiences with a High-Fidelity Wireless Building Energy Auditing Network. In *Proceedings of SenSys*, 2009.

[15] S. Kim, R. Fonseca, P. Dutta, A. Tavakoli, D. Culler, P. Levis, S. Shenker, and I. Stoica. Flush: A Reliable Bulk Transport Protocol for Multihop Wireless Networks. In *Proceedings of SenSys*, 2007.

[16] S. Kim, S. Pakzad, D. E. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Health Monitoring of Civil Infrastructures Using Wireless Sensor Networks. In *Proceedings of IPSN*, 2007.

[17] J. Ko, J. Eriksson, N. Tsiftes, S. Dawson-Haggerty, M. Durvy, J. Vasseur, A. Terzis, A. Dunkels, and D. Culler. Beyond Interoperability: Pushing the Performance of Sensor Network IP Stacks. In *Proceedings of SenSys*, 2011.

[18] O. Landsiedel, F. Ferrari, and M. Zimmerling. Chaos: Versatile and efficient all-to-all data sharing and in-network processing at scale. In *Proceedings of SenSys*, 2013.

[19] C.-J. M. Liang, N. B. Priyantha, , J. Liu, and A. Terzis. Surviving Wi-Fi Interference in Low Power ZigBee Networks. In *Proceedings of SenSys*, 2010.

[20] R. Musaloiu-E, C.-J. M. Liang, and A. Terzis. Koala: Ultra-Low Power Data Retrieval in Wireless Sensor Networks. In *Proceedings of IPSN*, 2008.

[21] R. Musaloiu-E, A. Terzis, K. Szlavecz, A. Szalay, J. Cogan, and J. Gray. Life Under Your Feet: A Wireless Soil Ecology Sensor Network. In *Proceedings of EmNets*, 2006.

[22] F. Osterlind and A. Dunkels. Approaching the Maximum 802.15.4 Multihop Throughput. In *Proceedings of HotEmNets*, 2008.

[23] J. Paek, K. Chintalapudi, R. Govindan, J. Caffrey, and S. Masri. A Wireless Sensor Network for Structural Health Monitoring: Performance and Experience. In *Proceedings of EmNetS-II*, 2005.

[24] M. Rahimi, R. Baer, O. I. Iroezi, J. C. Garcia, J. Warrior, D. Estrin, and M. Srivastava. Cyclops: In Situ Image Sensing and Interpretation in Wireless Sensor Networks. In *Proceedings of SenSys*, 2005.

[25] H. Rahul, H. Hassanieh, and D. Katabi. SourceSync: A Distributed Wireless Architecture for Exploiting Sender Diversity. In *Proceedings of ACM SIGCOMM*, 2010.

[26] B. Raman, K. Chebrolu, S. Bijwe, and V. Gabale. PIP: A Connection-Oriented, Multi-Hop, Multi-Channel TDMA-based MAC for High Throughput Bulk Transfer. In *Proceedings of SenSys*, 2010.

[27] N. Ramanathan, T. Schoellhammer, E. Kohler, K. Whitehouse, T. Harmon, and D. Estrin. Suelo: Human-Assisted Sensing for Exploratory Soil Monitoring Studies. In *Proceedings of SenSys*, 2009.

[28] L. Selavo, A. Wood, Q. Cao, T. Sookoor, H. Liu, A. Srinivasan, Y. Wu, W. Kang, J. Stankovic, D. Young, et al. Luster: Wireless Sensor Network for Environmental Research. In *Proceedings of SenSys*, 2007.

[29] K. Srinivasan, M. Jain, J. I. Choi, T. Azim, E. S. Kim, P. Levis, and B. Krishnamachari. The K-Factor: Inferring Protocol Performance Using Inter-link Reception Correlation. In *Proceedings of Mobicom*, 2010.

[30] G. Tolle, J. Polastre, R. Szewczyk, D. Culler, N. Turner, K. Tu, S. Burgess, T. Dawson, P. Buonadonna, D. Gay, et al. A Macroscope in the Redwoods. In *Proceedings of SenSys*, 2005.

[31] Y. Wang, Y. He, X. Mao, Y. Liu, Z. Huang, and X. yang Li. Exploiting Constructive Interference for Scalable Flooding in Wireless Networks. In *Proceedings of INFOCOM*, 2012.

[32] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh. Fidelity and Yield in a Volcano Monitoring Sensor Network. In *Proceedings of OSDI*, 2006.

[33] T. Zhu, Z. Zhong, T. He, and Z.-L. Zhang. Exploring Link Correlation for Efficient Flooding in Wireless Sensor Networks. In *Proceedings of NSDI*, 2010.

[34] M. Zimmerling, F. Ferrari, L. Mottola, and L. Thiele. On Modeling Low-Power Wireless Protocols Based on Synchronous Packet Transmissions. In *Proceedings of MASCOTS*, 2013.