USC CENG Technical Report
CENG-2014-01

BackIP: Backpressure Routing in IPv6-Based Wireless Sensor Networks

Srikanth Nori
Department of Computer Science
University of Southern California
snori@usc.edu

Suvil Deora
Department of Electrical Engineering
University of Southern California
deora@usc.edu

Bhaskar Krishnamachari
Department of Electrical Engineering
University of Southern California
bkrishna@usc.edu

ABSTRACT

Wireless sensor networks are increasingly being deployed in real-world applications ranging from energy monitoring to water-level measurement. To better integrate with existing network infrastructure, they are being designed to communicate using IPv6. The current de-facto standard for routing in IPv6-based sensor networks is the shortest-path-based RPL, developed by the IETF 6LoWPaN working group. This paper describes BackIP, an alternative routing protocol for data collection in IPv6-based wireless sensor networks that is based on the backpressure paradigm. In a backpressure-based protocol, routing decisions can be made on-the-fly on a per-packet basis by nodes based on the current locally observed state, and prior work has shown that they can offer superior throughput performance and responsiveness to dynamic conditions compared to shortest-path routing protocols. We discuss a number of design decisions that are needed to enable backpressure routing to work in a scalable and efficient manner with IPv6. We implement and evaluate the performance of this protocol on a TinyOS-based real wireless sensor network testbed.

Categories and Subject Descriptors

C.2.2 [**Computer Systems Organization**]: Network Protocols—*Routing protocols*

General Terms

System Design, Performance, Measurement, Experimentation

Keywords

Wireless Sensor Networks, 6LoWPAN, IPv6, Backpressure, Routing

1. INTRODUCTION

Wireless sensor networks are being deployed in practical environments such as energy monitoring setups for office buildings, smart-home meters, customer localization within retail spaces and asset tracking and surveillance in factories. Such deployments become far more valuable to their end users if they integrate better with existing network infrastructure on location. This need for integration has motivated the deployment and use of IPv6 within sensor networks.

The IETF 6LoWPAN working group [1] has been working towards standardizing protocols for sensor networks that utilize IPv6. To this end, an IPv6 compatible packet format was defined in RFC4944 [7]. The 6LoWPAN WG also defines the Routing Protocol for Low-Power and Lossy Networks (RPL) [9] - a framework and protocol for implementing routing for various objective functions.

The Collection Tree Protocol (CTP) [3] operates by constructing trees from sensor nodes towards the sink. CTP minimizes ETX (or an ETX-dependent metric that includes bandwidth information) to locate tree parent nodes. The core concept of RPL centers around building Destination Oriented Directed Acyclic Graphs (DODAGs, or trees) from source to destination, for various objective functions (OF). Thus, RPL with an OF of minimizing ETX operates very similar to CTP. The quasi-static trees constructed are periodically updated to adapt to changing link conditions, but there is still a possibility of nodes operating on outdated trees (especially when news of bad links has not propagated).

The Backpressure Collection Protocol (BCP) [6] is an alternative collection mechanism for sensor networks. BCP uses backpressure values to make per-packet routing decisions, as opposed to the quasi-static trees set up within CTP. This allows BCP to react better to dynamically varying link conditions.

In this paper we describe the design of a system that implements backpressure routing on an IPv6 sensor network for data collection. Backpressure routing is based on the concept of *Utility Optimal Lyapunov Networking* as defined in [8]. In backpressure protocols each node maintains a queue of outgoing packets, and shares the queue size information with its neighbors. A node decides the best next hop for a packet based on three factors: queue size differential, transmission rate, and the estimated number of transmissions (ETX) to each

This work has been supported in part by the U.S. National Science Foundation, under award 1049541.

neighbor. This routing decision is made for every single packet transmitted (as the factors keep changing over time). This approach allows nodes to react better to varying network conditions. BCP also includes a practical implementation of such routing for sensor networks, and we use this as a base for our design. However BCP uses a custom packet format and implements a complete collection mechanism, and not one that can integrate with IPv6 based networks. We implement BackIP as a separate routing block as a part of the IP stack.

We outline the challenges we faced in designing this system. The challenges arise due to constraints imposed by the design of IP, the software architecture, and the hardware capabilities of sensor motes. We address these challenges to fit in with the IPv6 architecture.

Specifically, we design a method to efficiently integrate backpressure queues into the IP stack. We develop a new queue-based adaptive beaconing mechanism to reduce network bandwidth as well as reduce the CPU load on the nodes. We adapt NULL packets, a special control packet, to integrate with the IP architecture. We design standard-compliant ways to disseminate queue information across the network. And finally we implement the protocol so as to efficiently coordinate data and control information across software layers within the node.

We implement our protocol on TinyOS [5] using the Berkeley IP implementation for low-power networks (BLIP) [2] and analyze its performance in detail. (We use the latest released version at the time of writing this paper: TinyOS 2.1.2). We also describe the results from our testing after we deployed an implementation of our protocol on a 40-node testbed. Some studies have been done in the past to study routing performance. Ko et al [4] have specifically studied the performance of TinyRPL - the RPL implementation on TinyOS - in comparison with CTP. Along similar lines in this paper we compare the performance of TinyRPL with BackIP.

Our design has multiple goals that we address in detail in subsequent sections:

- **Operate within resource constraints**: The nodes are low power devices with low CPU power, small memory capacity and limited capacity batteries. These networks also use the IEEE 802.15.4 standard for radio communication and these channels offer limited bandwidth.
- **Conform to existing standards**: The protocol data must conform to the IPv6 packet/addressing format as well as 802.15.4 MAC packet format
- **Adhere to IP architecture**: The protocol must operate within the IP layering design. Further we implement our protocol to integrate optimally with the BLIP implementation in TinyOS.

This work is a start towards a complete IPv6 routing protocol based on backpressure. Since a primary use case for many sensor networks is data collection, we first design the routing protocol for many-to-one operation with sufficient design consideration to allow for any-to-any transmission in the future.

2. BACKGROUND INFORMATION

We analyze the design and implementation of IP to propose an architecture for BackIP. In this section we first discuss some background information, which we then use to describe the design and implementation of BackIP.

2.1 Overview of Backpressure

Backpressure routing techniques have been explained in detail in BCP [6]; this section presents a high-level overview. Each node maintains a queue of outgoing packets and the queue occupancy defines the 'backpressure' of a node. The node services the queue and transmits packets by choosing the best neighbor for each packet. A node that is successfully able to transmit packets quickly thus empties its queue and exerts a low backpressure, indicating good link quality. A node that is unable to transmit packets fills up its queue and exerts a higher backpressure, indicating poor link quality.

To achieve this, in BCP each node maintains a table of information about its neighbors. Each entry in the table contains the queue size, estimated number of transmissions (ETX) and transmission rate to each neighbor. At every instant that a node needs to transmit a packet it selects the best neighbor from the information in the table based on the queue occupancy differential and network parameters. The node calculates backpressure weight for each neighbor and chooses the best neighbor as the next hop. This formula used for calculating the weight is reproduced here for convenience:

$$w_{i,j} = (\Delta Q_{i,j} - V \cdot \Theta_{i \to j}) \cdot R_{i \to j}$$

Here, $\Delta Q_{i,j} = Q_i - Q_j$ is the difference in queue sizes (backpressures) of nodes i and j, $R_{i,j}$ is the current known link rate, and $\Theta_{i \to j}$ is a penalty that depends upon the penalty/utility function. V is a tradeoff between queue occupancy and penalty minimization.

Node i chooses the next hop as the node j with the *highest* positive value of $w_{i,j}$ from among all its neighbors. If no node has a positive value of $w_{i,j}$ then the packet is held and forwarded only when a good neighbor eventually becomes available.

Maintaining up-to-date information about neighbor backpressure is critical to ensure good routing performance.

2.2 Overview of BLIP

BLIP provides an implementation of RFC4944 for TinyOS. Specifically, BLIP provides support for plug-

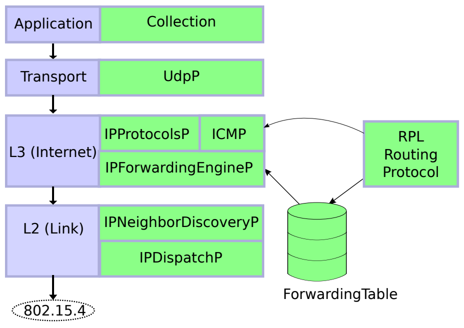

Figure 1: BLIP components

ging in an external routing protocol. Figure 1 has a diagram of the layers. Note that for optimal performance and reduced memory usage the stack implements 'zero-copy' semantics. Briefly, the functions of each layer are:

- *UdpP*- UDP transmission/reception
- *IPProtocolsP*- Addressing, IP header construction /parsing, ICMP
- *IPForwardingEngineP*- End-to-end routing. Forwarding table with longest prefix match
- *IPNeighborDiscoveryP*- L3/L2 address translation.
- *IPDispatchP*- Header compression, fragmentation and framing in 802.15.4 format
- *Routing Protocol*- Routing specific functionality. e.g. RPL DODAG construction is implemented here.

2.3 Overview of RPL

We compare the performance of BackIP with RPL. In RPL, each node maintains a route to the data sink by locating the 'parent' node for itself in the DODAG, thus creating trees rooted at the sink.

RPL is very flexible in that it allows nodes to choose the parent node for routing based on a generic objective function (OF) that operates on any routing metric (e.g. ETX). RPL also allows multiple DODAGs with different OFs to coexist on the same network. TinyRPL is the implementation of RPL for TinyOS. After calculating the parent TinyRPL installs routes using the forwarding module within BLIP.

3. BACKIP DESIGN

3.1 Design overview

3.1.1 Operate within resource constraints

The core of backpressure algorithms is based on queue differentials. It is necessary to maintain a queue of outgoing packets at some layer of the node software. The 'simplest' location would be to place the queue at the IP Protocols layer, but here packet headers alone could exceed 40 bytes per packet. For quality backpressure operation the queue needs to be sufficiently large, so we must optimize the size of each queue element. Thus we queue up packets *after* header compression, so we maintain the packet queue at the south end of the BLIP stack at the MAC interface. This is a critical design decision that influences a large number of subsequent design choices.

3.1.2 Conform to existing standards

Backpressure operation requires that nodes maintain updated backpressure info of all neighbors. This is achieved in three ways that require special control information on the network - periodic beaconing, per-packet headers, and NULL packets. We design our protocol so that it does not break interoperability with 802.15.4 and IPv6.

Queue-based adaptive beacons.
Nodes inform their neighbors of their backpressure by periodically sending beacons that carry only local backpressure value. To maintain IP interoperability, beacons are sent as UDP packets on a reserved port. We create a novel technique to reduce the beaconing load, which is described in section 3.3.1. Beacons are transmitted as IPv6 link-local broadcasts (i.e. sent to `ff02::1`) on a reserved UDP port. Broadcasts are not forwarded across the network - they only reach the immediate neighbors of a particular node.

Per-packet headers.
Nodes also send their local backpressure in all *data* packets they generate or forward. This lets backpressure info piggyback on actual data to keep neighbors more up-to-date. For the per-packet data we could potentially design in a custom MAC protocol format or a custom IPv6 payload, but that would break interoperability. Thus we choose to place queue occupancy information in the IPv6 hop-by-hop extension header for each packet(section 3.4). When a node receives any packet it inspects the hop-by-hop header. If the BackIP header is found (we use a reserved header type during development) the backpressure value in the neighbor table is updated. The node updates the value with its own backpressure before forwarding the packet.

NULL packets/Virtual Queues.
BCP makes use of virtual queues, or floating queues, to reduce the amount of data stored on each node. Virtual queues in BCP are described in detail in section 3.3 of [6]. We use a separate UDP port to handle NULL packets, which are sent from sources all the way to the sink, where they are discarded after processing header contents. NULL packets contain no data, but they have the hop-by-hop header.

3.1.3 Adhere to IP architecture

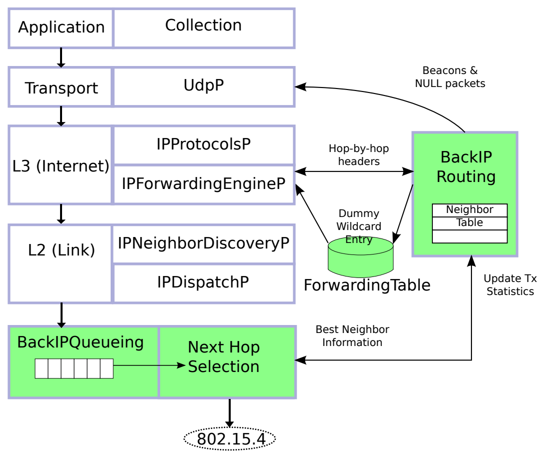

Figure 2: BackIP Integration

BackIP operates at a similar block as RPL thus maintaining routing as a separate function. However, a major change is that the selection of the next hop is now done entirely at the south end of the BLIP stack. So the role of the forwarding table is significantly reduced. We write a single wildcard entry into the forwarding table to push through all data directly into the backpressure queuing block. This change is necessitated by the first design goal - we want to queue compressed data, but we also want to be able to change the destination of a data packet on the fly (even after compression). To reduce CPU load and maintain IP layering we also avoid the use of 'snoop' to keep neighbor information updated.

3.1.4 Summary

Figure 2 shows the resultant architecture for BackIP. Layers that differ from the base architecture in Figure 1 are marked in green.

3.2 BackIP operation

When a BackIP node starts operation it writes a default entry into the forwarding table. When nodes transmit data they write their local backpressure into the hop-by-hop headers. The complete headers are compressed by BLIP and data is then copied into BackIP's queue (which breaks the zero-copy semantics of BLIP; This results in higher memory utilization but, as we show in subsequent sections, this is a worthwhile trade-off to improve throughput). If the queue is full, the oldest element in the queue is discarded and virtual queue counter is incremented.

A separate task within the BackIPQueueing block runs to service the queue. The queue is operated in a Last-In-First-Out fashion, so the task picks out the element that was inserted most recently. If the data queue is empty but the virtual queue counter is non-zero a NULL packet is sent (and virtual counter decremented), otherwise a data packet is available and sent.

The task looks through the BackIP neighbor table and selects the best neighbor for a packet. This choice is made based on the backpressure exerted by all its neighbors as well as current network conditions of all their neighbors. If a good neighbor is found the destination MAC address of the packet (See section 3.4) is overwritten with the neighbor's MAC address, and the packet is transmitted. If a good neighbor is not found the packet is returned to the top of the queue and retransmission is scheduled for a later point in time. Beacons are triggered if necessary.

BackIP requires link-layer acknowledgments. If a packet is successfully acknowledged the ETX and transmission rate for this neighbor are updated immediately. If not, BackIP reattempts transmission up to 5 times (configurable), after which it pushes the packet to the end of the queue to be serviced later and penalizes the link.

3.3 Novel Contributions

Besides providing the first such implementation of dynamic backpressure routing over IP, BackIP provides the following novel design contributions:

3.3.1 Queue-based adaptive Beaconing

BackIP relies on beacons to disseminate backpressure information. Beaconing too rarely would cause network information to be stale, whereas beaconing too frequently would increase the CPU usage and increase the network overhead. Thus we need to find a balance in the frequency of beaconing.

To this end we rely on a queue-based adaptive beaconing technique. The aim here is to ensure that network information is updated whenever necessary. We adopt a straightforward approach to this problem: Whenever the queue size in a node changes over a threshold the node generates a beacon with the updated information.

When a node joins the network it sends a special "request" beacon periodically until it receives a response from at least one neighbor. When a node receives a request beacon it rapidly responds with a regular beacon of its own.

This approach has multiple advantages over naive periodic forwarding:

- Only nodes that are transmitting data, thus changing their queue sizes, generate beacons.
- New nodes beacon rapidly only until they join the network. Beyond that they only beacon as needed
- The threshold for beaconing can be modified based on the number of neighbors in a network to scale well.

Setting the threshold too low causes the network to flood with beacons. Setting this too high causes nodes

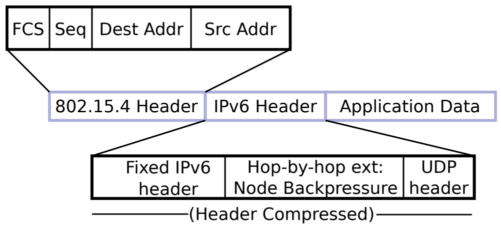

Figure 3: Frame Format

Link-Layer Address	Latest Backpressure	EWMA of ETX	EWMA of Tx rate
0x22	2	7.22	140
0x13	3	1.22	210
...	...	...	...

Table 1: Neighbor table

to react less frequently to bad conditions. It is empirically set to 3 in our testing. The idle beacon interval (Set empirically to 500ms) is created to ensure that when a node has no neighbors it generates beacons at this fixed interval. The data sink also generates beacons at this interval (since its backpressure never varies).

3.3.2 Operation without snoop

BCP relies on nodes 'snooping' on all data intercepted and uses these to keep the neighbor table updated. In case of BackIP if nodes were to snoop then every node would need to decompress packets not intended for it. The CPU overload for this would be unacceptable, so We designed BackIP to operate without snoop. We eliminate the use of snoop and rely on beacons and per-packet headers to maintain updated network state.

3.4 Data Formats

Figure 3 has the complete packet format. Table 1 shows the structure of BackIP neighbor table with sample entries.

4. EXPERIMENTAL RESULTS

4.1 Test Methodology

We test the protocol on a 40 node testbed of Tmote Sky devices (on Tutornet at the University of Southern California). Transmit power is set to -25dBm and all devices use channel 26.

Total data packet size is 40 bytes for BackIP - this includes 802.15.4 headers, compressed IP headers (including BackIP extension header) and 4 bytes of data. Packets for RPL are also of the same size, with an RPL extension header in place of BackIP. Statistics are collected via the serial interface on the motes.

4.2 Test Results

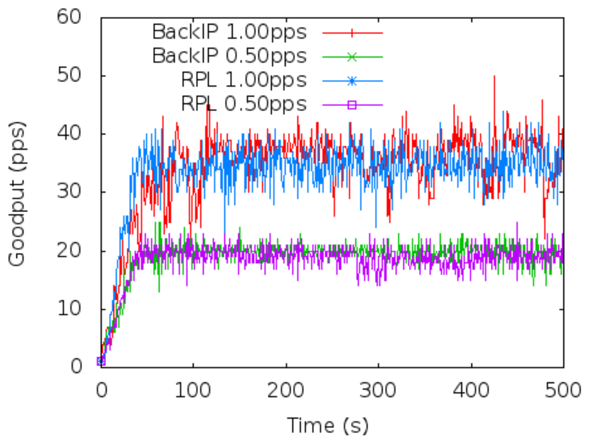

Figure 4: Goodput at root on testbed over time.

For all these experiments we first activate the root and then activate senders sequentially (at 1s intervals). The nodes send data periodically at configurable periods.

4.2.1 Goodput of the network

The goodput (eliminating duplicates and control packets) is measured at the root. All 40 senders are configured to generate packets periodically. Table 2 shows the total packet reception ratio at the root for different transmit frequencies for one run of each test. We see that RPL and BackIP have similar performances below 1 packet-per-second (pps). As the network load approaches 1 pps BackIP has a higher reception rate than RPL.

Figure 4 plots the goodput at the root over time for two offered rates for BackIP and RPL. The throughput builds steadily until all nodes are turned on and then stabilizes to the expected value accounting for the PRR. We can see that, at higher frequency of transmission, BackIP gets a higher PRR.

4.2.2 Delay and re-ordering

Due to the queuing inherent to BackIP it is possible that delays can increase. We study the delay performance to find the actual impact. Figure 5 plots the CDF of end-to-end delay for all packets at various send rates. We see that the median delay stays well within the 50ms range, but there are occasional instances of very high delay where packets are held in queues for long lengths of time, and this is independent of send rate. We can also see that RPL shows low delay for 1pps setting, this is because when congestion collapse occurs in a static routing tree only the nodes near the sink are able to get packets to the destination (with low delay), but this is absent in BackIP as nodes are equally affected by collapse.

Queuing could also lead to packet reordering. We

Offered Rate (per node)	BackIP	RPL
0.25pps	96.39%	98.49%
0.5pps	97.5%	95.86%
0.75pps	96.53%	94.54%
0.875pps	89.97%	88.18%
1pps	88.92%	61.86%

Table 2: PRR vs offered rate

Reorder extent	BackIP 1pps	RPL 1pps	BackIP 0.5pps	RPL 0.5pps
1	0.938	0.871	0.983	0.968
2	0.029	0.089	0.008	0.027
3	0.008	0.023	0.002	0.003
4	0.008	0.009	0.001	0.001
4+	0.017	0.008	0.006	0.001

Table 3: Reordering extent PMF

measure reordering by sending packets with sequence numbers from each node and calculating the extent of reordering observed at the root. We see that there are rare cases where there is any large reordering observed, and this is also independent of send rate. The PMF of observed absolute reordering extent for BackIP and RPL at two send rates is given in Table 3.

5. CONCLUSION AND FUTURE WORK

In this paper we presented the first steps towards a complete backpressure based routing protocol for IPv6 based wireless sensor networks by creating a data collection protocol. We presented the challenges inherent in converting a queue-based protocol to fit within the IP framework and explained our proposal and decisions in designing this architecture. We implemented this protocol using TinyOS and deployed it on a 40 node testbed. We showed empirically that, for data collection, BackIP performs at least as good as tree-based routing mechanisms for similar workloads, and that it provides improved performance at higher loads. We maintained IP compatibility throughout the design and implementation.

We designed the architecture with specific extensions in mind - our target is to create a full-featured protocol including any-to-any and one-to-many routing using backpressure. The BackIP queueing layer is designed to be able to handle multiple queue gradients across multiple pairs of neighbors across the network (and not just many-to-one). We also plan to study interoperability with other routing protocols in scenarios where it may be necessary to have other routing protocols running on the network. From our experience constructing BackIP we believe that it can provide a useful alternative to RPL for specific kinds of networks.

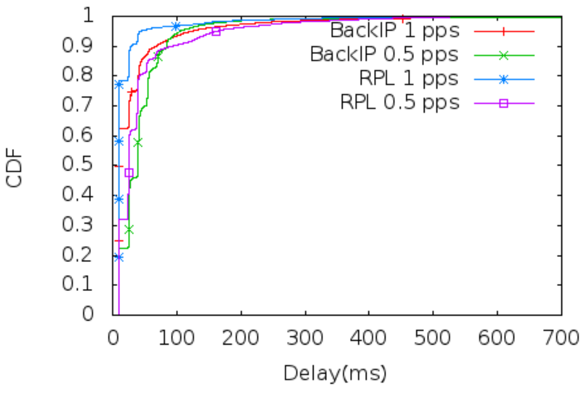

Figure 5: Delay CDF on testbed

6. REFERENCES

[1] IPv6 over Low power WPAN - Charter for Working Group. http://datatracker.ietf.org/doc/charter-ietf-6lowpan/.
[2] S. Dawson-Haggerty. Berkeley ip implementation for low-power networks, May 2010. http://smote.cs.berkeley.edu:8000/tracenv/wiki/blip.
[3] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. SenSys 2009. ACM.
[4] J. Ko, S. Dawson-Haggerty, O. Gnawali, D. Culler, and A. Terzis. Evaluating the performance of rpl and 6lowpan in tinyos. In *Proceedings of Extending the Internet to Low power and Lossy Networks (IP+SN 2011)*, April 2011.
[5] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, et al. Tinyos: An operating system for sensor networks. In *Ambient intelligence*. Springer, 2005.
[6] S. Moeller, A. Sridharan, B. Krishnamachari, and O. Gnawali. Routing without routes: the backpressure collection protocol. IPSN 2010. ACM.
[7] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944 (Proposed Standard), Sept. 2007. http://www.ietf.org/rfc/rfc4944.txt.
[8] M. J. Neely. Intelligent packet dropping for optimal energy-delay tradeoffs in wireless downlinks. *Automatic Control, IEEE Transactions on*, 2009.
[9] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. RFC 6550 (Proposed Standard), Mar. 2012. http://www.ietf.org/rfc/rfc6550.txt.