

PSFQ++ - A Source-to-Sink Reliable Transport

Max Baker

Midterm, E6906 – Sensor Networks – Fall 2004, Prof. Andrew Campbell

1. Introduction

Direct diffusion[1] provides a data-centric dissemination paradigm for wireless sensor networks. This paradigm accounts for the high packet loss nature of these networks, as well as the application-level tolerance of data loss. Directed diffusion introduces the notion of a one-to-many relationship between a data requester (sink) and data providers (source) where data is disseminated through the sensor network on its path towards the sink. The sink does not use or require the knowledge of which source the data is from. However, as the applications for sensor networks evolve they are beginning to require a need for end-to-end reliability between sinks and sources. Take for example a wireless sensor network used for target recognition. A source may typically provide information about the amount of targets seen during a given time frame. This data is not crucial and loss can be tolerated. If then the source senses a target that is deemed important, this information will want to be *reliably* transferred to the sink.

A new method needs to be developed to allow reliability of data between source and sink. This paper seeks to combine the work of two existing reliable transport protocols: PSFQ[2] and RMST[3] into a new reliable transport mechanism: PSFQ++.

2. Related Work

2.1. PSFQ

Pump Slowly, Fetch Quickly (PSFQ) by Wan, et al. is the basis for PSFQ++. PSFQ provides reliable transport of data from a sink to all nodes in a sensor network over direct diffusion [1]. PSFQ is designed to push a code image out to remote sensor nodes from one connected, “user node”. The protocol leverages the fact that each sensor needs to keep the full file image in order to retask itself. A hop-by-hop recovery scheme of missing file fragments is used, as it is shown that providing end-to-end recovery over a wireless multi-hop link has an exponential probability of loss and does not scale [2]. Retransmission of missing packets is based on a NACK system whereby the missing packets are requested from neighboring nodes. Since the file transfer is targeted to all nodes on the network, PSFQ also takes advantage of the broadcast nature of a wireless network. Without a unique destination, a routing path is not required for distribution of the file and the forwarding of NACKs. Neighboring nodes are able to suppress flooding of the network by overhearing missing packet requests and the resulting responses.

In order to extend PSFQ to provide for reliable source to sink transport a few things are modified. First, the protocol cannot work with the assumption that every node will receive and store each transmission in a broadcast manner. Second, the protocol has to deal with more information passing through its data cache, as a node may be routing more than one reliable flow.

2.2. RMST

Another reliable transport algorithm proposed for sensor networks is the Reliable Multi-Segment Transport (RMST) protocol by Stann, et al. Like PSFQ, RMST is built on top of directed diffusion. RMST is designed to provide reliable transport of data from source to sink using changes made in the MAC, transport, and application layers. RMST provides guaranteed delivery of file fragments without the need for or guarantee of in-order delivery. Data is both broadcast and unicast – route establishment using directed diffusion is broadcast, while data flows over unicast hops.

RMST uses the same notion of hop-by-hop selective request NACK for loss detection. Repair requests are sent to immediate neighbors. If the data is not found in the neighboring node's data cache, the NACK are forwarded up the routing tree towards the source. In order to service the request for a reliable flow, the notion of a backchannel is introduced. A reverse route from the source to the sink is created from the route information developed by directed diffusion.

3. Algorithm

PSFQ++ builds upon the three existing functions of PSFQ: pump, fetch, and report. On top of these functions I add the notion of a backchannel as in RMST. The backchannel is the route from source to sink that is constructed using the inverse of the route established by direct diffusion. The pump operation for source to sink reliable communication is unicast over this backchannel, while the fetch operations are broadcast. In addition to the three existing functions, PSFQ++ adds two ACK messages: Sector ACK and Report ACK. In this section I explore the operation of the transport, and in the next section I describe the management of cache data.

3.1. Reliable Report

When a source node wants to initiate a reliable flow with the sink it first sends out a *Reliable Report* message to the sink. This report mechanism is the same as the one detailed in PSFQ, except that an additional header field is added that contains the file size in bytes. From this message each node along the gradient, as well as the sink, can determine the number of fragments and sectors in the file. As this report message makes its way back to the sink, each node piggybacks its own report information on the message. The source will continue to send *Reliable Reports* until it receives a *Report ACK* from the sink.

3.2. Pump Very Slowly

At the same time as a source sends out a *Reliable Report*, it begins pumping data towards the sink using the pump mechanism of PSFQ. All Pump operations are unicast up the backchannel. An extra bit is added in the header field of the data to specify that the data is *exploratory data* and that the sink has not yet ACK'd the *Reliable Report*. During this phase the T_{\min} and T_{\max} are raised for pump operations in order to make the source pump very slowly.

There is a tradeoff between the wasting of resources in intermediate nodes and the increase of file transfer latency during the time the reliable flow between source and sink is being initiated. The network should not be idle while waiting for the Report/ACK phase between source and sink to complete. Data begins to be slowly cached along the route to the source

during this slow-pump phase. I call this *priming the pump*.

3.3. Report ACK

When the sink receives a *Reliable Report* from a source, a reliable flow is initiated. The sink then sends an ACK of this report back to the source, via directed diffusion. The sink counts up the number of nodes seen in the report and includes this information in the *Report ACK*. The sink will continue to send *Report ACKs* until data is received from the source that does not have the *exploratory flag* set in the header. Reception of data without the exploratory flag means the source has received the *Report ACK*.

3.4. Pump Slowly

Once the source receives the *Report ACK* from the sink it does two things. First, it removes the exploratory flag set to signal to the sink that the ACK was received. Second, it begins pumping faster. T_{\min} and T_{\max} are reduced in order to speed up pump operation, now that a reliable flow has been established. In the *Report ACK*, the source receives an approximation of how many hops are between it and the sink. This is only an approximation because of the volatile and dynamic nature of the routing in direct diffusion. This integer can be used to tune the T_{\min} and T_{\max} settings. The fewer hops between the sink and source, the faster the pump operations. As the number of hops grows, the protocol accounts for the extra probability of fragment loss and resulting fetch operations along the path by pumping slower.

3.5. Fetch Quickly

When a sequence number gap is detected on a node, the node will send a NACK upstream. This NACK is broadcast so that neighbors receive the NACK will be able to send the node the missing fragment if they have it in their cache as is explained in *non-gradient neighbors* below. As no traffic is forwarded until the hole is filled, this fetch operation is performed aggressively. Retransmissions take priority over new traffic.

4. Data Cache Management

In the original PSFQ there is one data cache on each node to hold the new binary image. Every node needs the same information so only one cache per node is required. In PSFQ++ each node needs to be able to keep track of data coming from multiple source-to-sink flows. The data cache cannot possibly hold the entire file for each flow.

Each file is broken up into sectors, and the data cache of each node is broken up into pages. The size of the cache pages is equal to the size of the file sectors, as is shown in Figure 1.

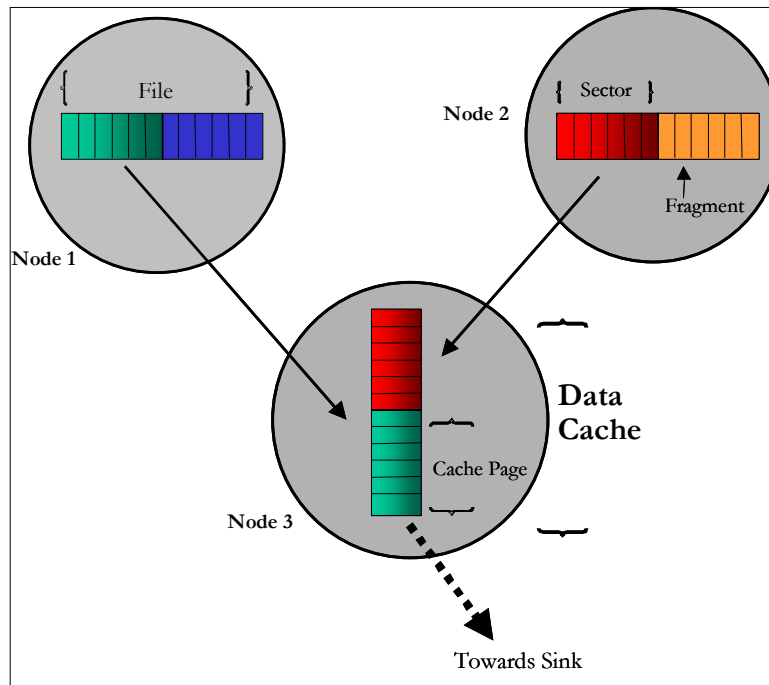


Figure 1 - File sectors occupy cache pages

4.1. Sectors

In order to control the data cache for each node, each file is broken up into sectors. Each sector contains an integer number of file fragments defined as the *sector size*. The final sector of a file, as well as the final fragment may not be totally full. The total number of sectors in a file is calculated as:

$$\text{number of sectors} = \text{ceil}(\text{number of fragments} / \text{sector size})$$

The sector number for a given fragment is zero-indexed and defined as:

$$\text{sector number} = \text{floor}(\text{fragment number} / \text{sector size})$$

4.2. Cache Paging

The data cache on a routing node has to support multiple source-to-sink reliable flows coming from its neighbors. The cache is broken up into pages that are the same size as sectors. A cache index is kept in local store that holds five fields per each cache page: the source node, the sector number, a complete flag, a local flag, and a LRU number. Cache pages may be swapped from permanent store to local store (EEPROM to SRAM) for power savings and processing speed, depending on the architecture of the nodes.

The complete flag is set when the node senses that the sector has successfully been transferred up the gradient, as is described below. If the data in the cache page was sent to the node unicast, the node is part of the gradient and the local flag is set. If the node overheard the data from a neighboring node, the local flag is not set.

4.3. Sector ACKs

After the sink has received all the fragments in a sector it sends out a *Sector ACK*. This ACK allows nodes in the gradient to clear their corresponding cache page for that sector. It

is important to note that the data caches of the nodes closest to the sink will fill up faster. However they will also receive the *Sector ACKs* first and will be able to clear cache pages quicker. Intermediate nodes forward the *Sector ACKs* up the gradient as they are received. *Sector ACKs* are not sent reliably, so the sink may send more than one.

4.4. Cache Page Expiration

An explicit *Sector ACK* allows a node to mark the corresponding page in its cache as complete. However, the *Sector ACK* messages are not sent reliably, so other mechanisms must be in place to allow for page expiration.

The first mechanism used by PSFQ++ is a last-recently-used (LRU) scheme. Every time data is added or requested from a cache page, its LRU is decremented by one down to a minimum of zero. Every other cache page is incremented by one at the same time. The pages with the highest LRU are expired first, given the rules below.

The second mechanism used for cache page expiration is an implicit ACK. If a downstream node or sink NACKs for a fragment in a sector that is greater than the current sector, the current sector can be marked as complete. Because of the in-order delivery guarantee of PSFQ, if a node NACKs for a fragment then it is implied that it has successfully received all the other fragments up to that one.

If a fragment in a new sector not previously seen reaches the node, the following algorithm is used to determine which cache page will be expired. If there are any pages marked complete, the page with the highest LRU is used. If no pages are marked complete, then the non-local page with the highest LRU is used. If there are no non-local and no complete pages, then finally the page with the highest LRU is used. The final case is important to note, because it is possible that a sector that is not complete can be cleared from the cache. I call this a *page wipe*. A *page wipe* will cause the node to NACK for all the fragments up to the current one in that sector. Clearing a cache page before the sector is complete is very detrimental to the performance of PSFQ++. The sector size must be tuned so that sectors are transferred fast enough to prevent *page wipes*.

The reception of an *implicit ACK* by a node signifies that a sector has been transferred to the next node up stream. This is not the same as a *Sector ACK*, because the sector may still have more hops before it makes it to the sink. Because nodes can drop off the network and route changes can occur, a sector that has been marked as complete by an *implicit ACK* may be requested again later by a new node in the gradient. Completed cache pages are not automatically wiped, but are subject to the LRU scheme, so that the algorithm can recover quicker in this case. If a completed page is already been expired, the intermediate node will have to NACK the data from upstream.

4.5. Sector Size

Choosing the correct sector size will affect the performance of PSFQ++ the most. If too large a sector size is used, then a whole sector may be cleared from the cache before it is completely transferred and an intermediate node or sink will have to NACK towards the source in order to get the same fragments over again. If the sector size is chosen as too small, the cache index may become too large for local store or become too long to process

quick enough. Choosing the sector size will depend on the fragment size of the underlying protocol, the processing power of the mote, and the amount of storage that can be allotted to the data cache and cache index. Simulation would help find the optimal sector size for a given architecture.

4.6. Non-Gradient Nodes

PSFQ++ exploits the broadcast nature of the wireless medium. Although the data from source to sink is unicast, other neighboring nodes that are not part of the gradient can overhear traffic. If the neighboring node has room in its data cache, it stores overheard traffic in its data cache. Cache pages that belong to overheard data do not have the local flag set. When a node NACKs for a file fragment, it does so via a broadcast. Neighboring nodes, regardless of if they are in its gradient can return the data if they have it in their cache, using the algorithm setup in PSFQ. This allows for the *fetch quickly* paradigm of PSFQ. Another consequence of broadcasting NACKs is to allow the network to heal after route changes.

4.7. Future Data Cache Enhancements

Limitation on the size of the data cache can cause cache pages to expire too rapidly. The data caching mechanism could be improved by providing a mechanism so that every node in a gradient does not cache data for every flow. An approach whereby every other node in a gradient caches data for a flow could be used. If the different flows were staggered correctly in a gradient, then each node would only have to hold sectors for half as many flows. The drawback would be that fetch data would come from two hops away. This is only suitable to deep hop-to-hop networks and is not applicable to “shallow” networks.

5. Conclusion

In this paper I show how PSFQ can be extended with some of the ideas of RMST to allow for reliable transport of data from source to sink in a directed diffusion based wireless sensor network. This paper does not address issues such as fairness between nodes that are close to the sink and far from the sink. Another issue that warrants investigation is the concerns of power usage. Neighboring nodes that are not part of a gradient may decide to sleep through messages that are not meant for them. They would not suffer the energy costs of caching and serving the data for subsequent fetches of neighboring nodes. The techniques for data cache management and reliable transport presented here still need to be simulated and implemented in a sensor network in order to fine-tune the implementation of PSFQ++.

References

- [1] Intanagonwiwat, Govindan, Estrin, Heidemann, Silva, “Directed Diffusion for Wireless Sensor Networks”, IEEE/ACM TON Vol 11, No. 1, Feb. 2003.
- [2] Wan, Campbell, Krishnamurthy, “PSFQ: A Reliable Transport Protocol for Wireless Sensor Networks”, First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA 2002), Atlanta, September 28, 2002.
- [3] Stann and Heidemann, “RMST: Reliable Data Transport in Sensor Networks”, First IEEE International Workshop on Sensor Net Protocols and Applications (SNPA), April, 2003.