Reliable and Efficient Data Transfer in Wireless Sensor Networks via Out-of-Sequence Forwarding and Delayed Request for Missing Packets

Damayanti Datta and Sukhamay Kundu
Computer Science Dept, Louisiana State University
Baton Rouge, LA 70803, USA
{ddatta1@lsu.edu, kundu@csc.lsu.edu}

Abstract

We present a new protocol for lossless data transfer in wireless sensor networks using less time and fewer messages in comparison to the well-known protocol PSFQ [1]*, without compromising the reliability. The two key features for the better performance of our protocol are out-of-sequence packet forwarding and delayed request for missing packets.*

1. Introduction

Many applications in wireless sensor networks require reliable data transfer, e.g., instructions sent from a sink to a set of destination nodes to execute a new task. We typically partition the entire data set into a number of (bounded) fixed size packets p_i, $0 \le i < n$ ($n \ge 1$) and then send them packet by packet. The detection and recovery of missing packets are performed at the destination nodes and also at the intermediate nodes; the latter can reduce the total delivery time of all packets to the destination nodes and also decrease the number of messages. The reduction of number of messages is important due to the energy constraints of the sensors and the reduction of delivery time is important for time-critical applications.

A well-known transport layer reliable data transfer protocol, with near-zero tolerance for data loss, is Pump Slowly Fetch Quickly (PSFQ) [1]. It is a non-acknowledgment based method and uses in-sequence (IS) forwarding, where a node x sends a packet p_j only if it has previously sent each packet p_i, $i < j$, at least once. This tends to delay the delivery of p_j to a node and to increase the total delivery time. The Reliable Multi-Segment Transport protocol (RMST) [2] implements reliability at both MAC and transport layer, using a PSFQ-like method for the latter.

Our protocol is also non-acknowledgment based, but it uses out-of-sequence (OS) forwarding with delayed request for missing packets (RMP); we call it, in short, OSDRMP. It has the same reliability as PSFQ. In OS-forwarding, a node can send a packet p_j before sending one or more p_i, $i < j$. In particular, here a node x can have the complete set of n packets although none of its neighbors has the complete set; this is not possible in IS-forwarding. Also, a node x can get all the packets quicker than in IS-forwarding. A node x may have a missing packet p_i for two reasons: p_i has not been sent to x or each transmission of p_i to x from its neighbors has failed to reach x. We use a minimum delay of t_r time units from the time p_i is detected missing at x till x can send the first RMP for p_i to its neighbors; this allows enough time for p_i to reach x from some of its neighbors. The same delay t_r is used between two successive RMPs from x for a given p_i. This prevents too many unnecessary RMPs for p_i in case none of the neighbors of x currently has p_i.

2. Key concepts

2.1. Acknowledgment vs. non-acknowledgment

Both the acknowledgment (ACK) and non-acknowledgment ($NACK$) based methods use three kinds of messages: (1) Transmission of a packet p_i, (2) Acknowledgment of a received packet p_i in ACK-based method or Request for Missing Packet p_i in $NACK$-based method, and (3) Retransmission of a packet p_i. We denote them respectively by $T(p_i)$, $ACK(p_i)$, $RMP(p_i)$, and $RT(p_i)$.

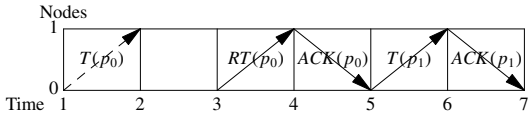
***ACK*-based method.** Here, a node waits for an $ACK(p_i)$ from the receiver for a minimum time period t_a after sending a $T(p_i)$. If it does not receive an $ACK(p_i)$, then it sends an $RT(p_i)$ and waits again for a time t_a for an $ACK(p_i)$, and the process continues till an $ACK(p_i)$ is received.

***NACK*-based method.** Here, with each $T(p_i)$ or $RT(p_i)$, we include the total number of packets n so that a node can detect the missing packets once it receives a packet and send RMPs. The delay t_r discussed in Section 1 applies to sending of RMPs. A node in this case does not wait for an $ACK(p_i)$ following a $T(p_i)$; instead, it sends an $RT(p_i)$ everytime it receives an $RMP(p_i)$. An $RMP(p_i)$ can be regarded as a negative acknowledgment. Unlike the ACK-based method, the NACK-based method cannot give 100% reliability if the successful transmission of a message has probability < 1.

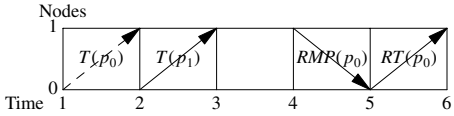
Example 1. Figs. 2(a)-(b) show the timing diagrams for the transmission of $n = 2$ packets $\{p_0, p_1\}$ from node 0 to node 1 in the network in Fig. 1(a) using *ACK* and *NACK*-based methods. We assume that only the first $T(p_0)$ is lost in both cases and all other messages are successful in the first attempt. The dashed lines in Fig. 2 show the lost transmissions. In Fig. 2(b), the receiving node 1 does not know the missing packet p_0 until it receives p_1. The example shows that the *NACK*-based method is better than the *ACK*-based method in terms of both the total delivery time and the total number of messages.

(a) A 2-node network. (b) A 4-node network.

Figure 1. Two simple networks.



(a) *ACK*-based method with $t_a = 1$.



(b) *NACK*-based method with $t_r = 1$.

Figure 2. An example of delivery of two packets $\{p_0, p_1\}$ in *ACK* and *NACK*-based methods.

2.1.1. Analysis of message efficiency for $\{p_0, p_1\}$

Assume as in above, that p_1 is successfully delivered to node 1 in one attempt. Let P be the probability of successful transmission of a message from one node to its neighbor. Hence, the probability that k transmissions are required for one message to be successfully sent from node 0 to node 1 is $P(1-\mathrm{P})^{k-1}$ and the expected number of transmissions is $1/P$. Let T_S and T_F denote respectively a successful and a failed transmission of a packet p_i; we use similar abbreviations for successful and failed *RT*, *RMP* and *ACK*.

In the *ACK*-based method, we have the following cases for the delivery of p_0 and $ACK(p_0)$: (1) a $T_S(p_0)$ from node 0 to 1 followed by an $ACK_S(p_0)$ or an $ACK_F(p_0)$ from node 1 to node 0, and (2) a $T_F(p_0)$ from node 0 to 1 (with no following $ACK(p_0)$). Except for the case of $T_S(p_0)$ followed by $ACK_S(p_0)$, there will be additional $RT(p_0)$s and $ACK(p_0)$s. Let $E(p_0)$ = the expected number of messages for delivery of p_0 and $ACK(p_0)$. Then, $E(p_0) = P^2.2 + P(1-P)[2 + E(p_0)] + (1-P)[1 + E(p_0)]$, and hence $E(p_0) = (1+P)/P^2$. Since we assume node 0 delivers p_1 in one attempt, the expected number of messages $E(p_1)$ in delivering p_1 and $ACK(p_1)$ is $1/P - 1$ less than that for p_0, i.e., $E(p_1) = (1+P^2)/P^2$. Hence, the expected number of messages for delivery of $\{p_0, p_1\}$ and their acknowledgments is $(2 + P + P^2)/P^2$.

In the *NACK*-based method, after $T_F(p_0)$ and $T_S(p_1)$ from node 0, node 1 detects that p_0 is missing and sends an $RMP(p_0)$. The computation of the expected number of messages $E'(p_0)$ for the delivery of p_0 is slightly more complex now. With probability P, p_0 is delivered via the initial $T_S(p_0)$. If the first $T(p_0)$ fails, then this will be followed by an average $1/P$ many $RMP(p_0)$ from node 1 to node 0 till the latter receives the request for p_0 and from that point on there will be an additional $E'(p_0)$ many messages for the delivery of p_0. Thus, $E'(p_0) = P.1 + (1-P)[1 + 1/P + E'(p_0)]$ and hence $E'(p_0) = 1/P^2$. The expected number of messages for delivery of $\{p_0, p_1\}$ here is $1 + 1/P^2$.

The *ACK*-based method therefore sends on the average $(1+P)/P^2$ extra messages. A similar analysis shows that the average number of messages received by nodes 0 and 1 is $2(1+P)/P$ for *ACK*-based method and $(1+P)/P$ for *NACK*-based method. Thus, *ACK*-based receives $(1+P)/P$ extra messages.

2.1.2. Analysis of message efficiency for $n \geq 2$

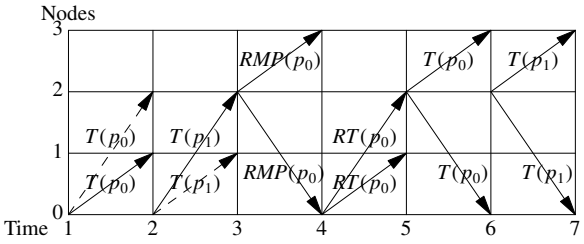
In the case of $n \geq 2$ packets, the probability that at least one packet has been successfully delivered in the first attempt is $Q_n = 1-(1-P)^n$ and assuming that this is the case the probability that $k \geq 1$ packets are delivered in the first attempt is $p(k) = \frac{1}{Q_n}\binom{n}{k}p^k(1-p)^{n-k}$. The expected number of messages sent in *ACK*-based method for the delivery of all n packets and their acknowledgments from node 0 to 1 is $\sum_{k\geq 1} p(k)[k.E(p_1) + (n-k)(1+E(p_0))]$. For the *NACK*-based method, the expected number of messages sent for the delivery of all n packets is $\sum_{k\geq 1} p(k)[k + (n-k)(1 + 1/P + E'(p_0))]$. This shows that the *ACK*-based method sends $n/(PQ_n)$ many extra messages, which can be very large when P is small; it is close to n when P is close to 1. We leave the analysis of the expected number of messages received for the general case to the reader.

2.2. Message priority in OS-forwarding

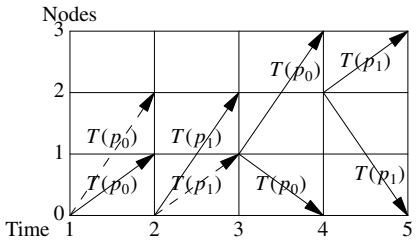
In OS, in general, the preferred priority order at a node among the message types is T > RT > RMP. Since a node can send a packet p_j without any

constraint on j, it should send the packets as soon as possible so that a destination node x may get different p_j from its different neighbors at the earliest. The preferred priority order in IS-forwarding (as in PSFQ [1]) is RMP > RT > T; here, it is better for a node to request a missing packet p_i because it cannot transmit p_j, $j > i$, before transmitting p_i. Unlike PSFQ, a node x does not delay the sending of Ts and RTs in OS-forwarding.

Example 2. Figs. 3(a)-(b) show the minimum time and number of messages for delivering $n = 2$ packets $\{p_0, p_1\}$ from node 0 to node 3 for the network in Fig. 1(b) using IS and OS. We assume that in both cases the only lost messages are $T(p_0)$ from node 0 to 2 and $T(p_1)$ from node 0 to 1, as indicated by the dashed lines. Here, we use $t_r = 0$ for simplicity. It shows how OS can achieve a better performance than IS.



(a) The use of 7 time units and 12 messages in IS.



(b) The use of 5 time units and 8 messages in OS.

Figure 3. Comparison of IS and OS forwarding.

2.3. NodeTTL and TTL in a message

Each node x which has received at least 1 message maintains a nodeTTL, which equals the maximum TTL (Time To Live) associated with those messages. Each message sent by x to its neighbors includes n, nodeTTL(x)−1, the message type (T, RT, or RMP), and one of the following: (1) the packet sequence number i and the data for p_i, for T and RT messages, and (2) the sequence number i of a missing packet p_i, for RMP messages. For the source node s, nodeTTL(s) is initialized to d and it does not change. We refer to the component nodeTTL(x)−1 in a message associated with p_i from x as packetTTL(p_i).

As a packet p_i travels away from the source s along various paths, the packetTTL(p_i) associated with the T and RT messages for p_i typically goes down by 1 with each step; it can also occasionally go up when it reaches a node that has previously received other messages along shorter paths. A node x with current nodeTTL(x) = 0 is considered a destination node, except that if at some later time nodeTTL(x) becomes greater than 0 then from that point onwards x remains permanently labeled as an intermediate node.

3. The New Protocol

3.1. Local data at a node

Each node x with nodeTTL(x) ≥ 0 has a Data Cache DC(x) = $\{p_i$: p_i received by $x\}$, a Transmission Queue TQ(x), a ReTransmission Queue RTQ(x) and a Request-for-Missing-Packet Queue RMPQ(x). At each node x, the following properties holds:

- DC(x) and RMPQ(x) are disjoint, TQ(x) and RTQ(x) are disjoint subsets of DC(x), and $|\text{DC}(x) \cup \text{RMPQ}(x)| = n$. None of TQ($x$), RTQ($x$) and RMPQ($x$) contains any duplicate item.
- The packets p_i in TQ(x) are ordered in the order of their arrival via Ts or RTs, and those in RTQ(x) are ordered in the order of their arrival via RMPs. RMPQ(x) is maintained as a circular list, ordered by the packet sequence numbers.

Example 3. Fig. 4 shows DC(x) of a node x for n = 9 and the *arrivalTime* of each p_i in DC; in particular, the packets arrived in the order p_1, p_4, p_5, p_0, and p_7. The packetTTL(p_i) is shown in parentheses next to each p_i in DC. Each of TQ(x), RTQ(x) and RMPQ(x) points to the first packet in the corresponding queue and the arrows from one packet to another show the sequence of packets in the queue. We also show the updates to nodeTTL(x) as each $p_i \in$ DC is received.

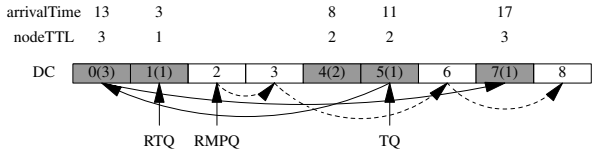


Figure 4. Illustration of DC, TQ, RTQ, RMPQ, and updating of nodeTTL at a node.

3.2. Message processing at a node

In one time unit, a node x can do one of the following: (1) receive a message from one of its neighbors and process it, (2) send a message to each of its neighbors, and (3) remain idle.

3.2.1. Nodes selected for message transmission

Two nodes can send a message at time t only if the hop-distance between them ≥ 3. The set of candidate nodes that are eligible to send a message at time t is given by $S'(t) = \{x$: TQ(x) $\cup$ RTQ(x) $\neq \emptyset$ or RMPQ(x) $\neq \emptyset$ and x can send $RMP(p_i)$ at time t for some $p_i \in$ RMPQ(x)$\}$. We can exclude a node x with nodeTTL(x) = 0 from $S'(t)$ for sending T and RT messages; this tends to reduce the number of messages slightly without significantly affecting the total delivery time in spite of the fact two destination nodes may be adjacent to each other and that a nodeTTL(x) may become positive at a future time.

Any protocol that prevents message collision at a node from two or more of its neighbors may be used to decide the group of nodes $S(t) \subseteq S'(t)$ that sends messages at time t. In our simulation (see Section 4), we select $S(t)$ as a random maximal subset of $S'(t)$ such that no two nodes in $S(t)$ are within hop-distance 2 of each other. A node in $S(t)$ transmits a message from one of its TQ, RTQ or RMPQ in that priority order. A node x sends RMP(p_i) for the first $p_i \in$ RMPQ(x) which satisfies t_r delay requirement. For each p_i, a node can send T(p_i) only once but it can send RMP(p_i) and RT(p_i) multiple times. At time t=0, $S'(0) = S(0) = \{s\}$. (Note that we cannot give priority to certain nodes x, say, based on nodeTTL(x) > 0 for selection in $S(t)$ since this would require a complex protocol in itself for the determination of $S(t)$.)

3.2.2. Processing of input

In addition to the update of nodeTTL(x), the data-cache and various queues at x are updated as follows when a node x receives an input. A node x processes an input message if and only if it is the first message to x with packetTTL ≥ 0 or nodeTTL(x) ≥ 0. If the first message with packetTTL ≥ 0 is a packet p_i (via $T(p_i)$ or $RT(p_i)$), x initializes DC(x) = $\{p_i\}$ = TQ(x), RTQ(x) = $\emptyset$, and RMPQ(x) = $\{p_j$: $j \neq i$ and $j \leq n-1\}$. If the first message with packetTTL ≥ 0 is an RMP(p_i), x initializes RMPQ(x) with p_0 and DC(x) = TQ(x) = RTQ(x) = $\emptyset$. When nodeTTL(x) ≥ 0, if the input is T(p_i) or RT(p_i) and $p_i \notin$ DC(x), then p_i is added to both DC(x) and TQ(x) and is removed from RMPQ(x), if necessary, RMPQ(x) is updated with all p_j where $p_j \notin$ DC(x)$\cup$RMPQ(x); if $p_i \in$ DC(x) then no other updates take place. If the input is RMP(p_i) then p_i is added to RTQ(x) provided $p_i \in$ DC(x) and $p_i \notin$ TQ(x)$\cup$RTQ(x); otherwise, no other updates occur.

4. Simulation

We simulate the OSDRMP and PSFQ protocols on the network in Fig. 5 for n = 10 for different source node s to evaluate their performance based on the total delivery time (delivTime) and the total number of messages sent and received (numMess). When a node x sends a message (T, RT, or RMP) to its neighbors, we count this as one message in numMess irrespective of the number of neighbors of x in order to reflect its impact on the energy consumption at x; each message received by x is also counted as one message in numMess. A simulation run is considered successful if at least one destination node x (i.e., nodeTTL(x) = 0) is found and each node x with nodeTTL(x) = 0 has |DC(x)| = n and TQ(x) = RTQ(x) = $\emptyset$; the condition TQ(x) = RTQ(x) = $\emptyset$ reduces the probability that there are other potential destinations nodes that have not received any p_i yet.

We consider two variations of RMPs described below. The second one applies only to OSDRMP.

(1) **Restricted RMP (rRMP):** Let m_x be the highest packet sequence number in DC(x) of a node x; we let $m_x = -1$ if DC(x) = $\emptyset$. Here, x can send an RMP(p_j) at time t only if $j < m_x$; the other restrictions that $p_j \in$ RMPQ(x) and x satisfies the delay requirement t_r still apply. If there is no $p_j \in$ RMPQ(x) with $j < m_x$ and RMPQ(x) $\neq \emptyset$ (i.e., $m_x < n-1$), then x can send RMP(p_j) for $j = m_x + 1$.

(2) **Unrestricted RMP (urRMP):** This is the normal case without the restrictions in **rRMP**.

We compute the minimum delay t_r for a node x according to t_r = degree(x)×RMPdelayFactor, where degree(x) is the number of neighbors of x and RMPdelayFactor is a constant = δ. A time point t is considered an *idleTime* if $S'(t) = \emptyset$.

We show below the results of simulation of OSDRMP (both rRMP and urRMP) and PSFQ for the network in Fig. 5 for P = 0.3, 0.6 and 0.9 using different RMPdelayFactor. We also show the results for the same network using s = 0 and TTL = 14.

4.1. Simulation results for s = 40 and TTL = 5

Figs. 6(a)-(f) show the variation of delivTime and numMess with RMPdelayFactor for OSDRMP(rRMP and urRMP) and PSFQ at P = 0.3, 0.6 and 0.9, s = 40 and TTL = 5. For higher P, we use a larger range of RMPdelayFactor to better represent the trend of variation of delivTime and numMess.

Both versions of OSDRMP perform better than PSFQ in terms of both delivTime and numMess when P = 0.6 and 0.9. For P = 0.3, numMess for PSFQ are

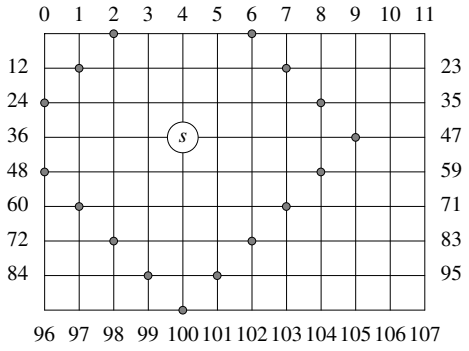


Figure 5. A 12x9 grid network; the intended destination nodes are marked for s = 40 and TTL = 5.

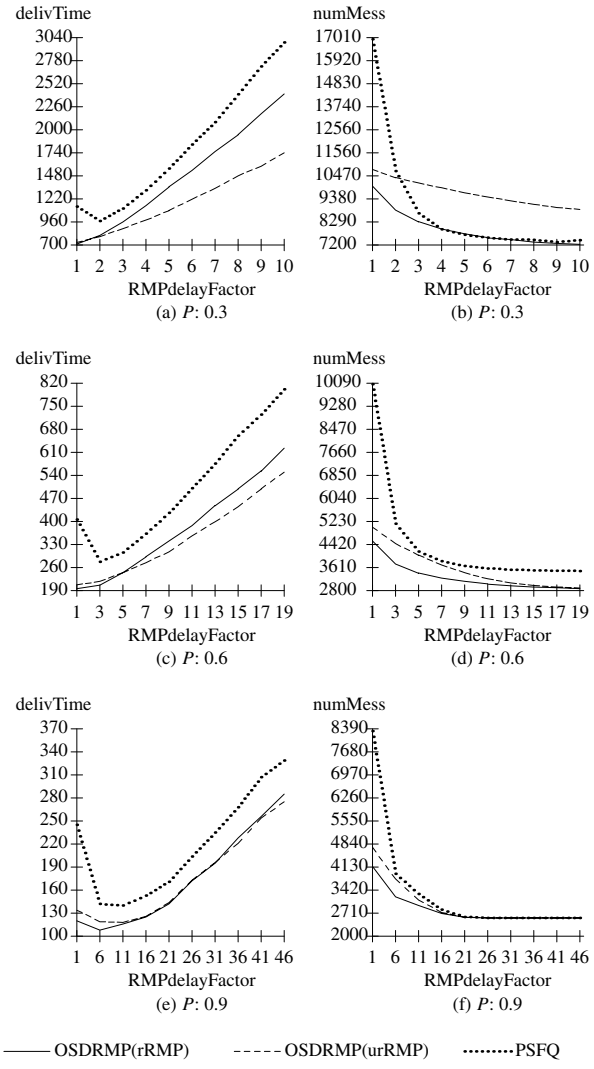


Figure 6. Comparison of OSDRMP and PSFQ for the network in Fig. 5.

initially higher than that for OSDRMP(urRMP) but as RMPdelayFactor increases, PSFQ performs better than OSDRMP(urRMP) but this comes only with a very large increase in delivery time.

4.1.1. Analysis: delivTime vs. RMPdelayFactor

Consider Figs. 6(a), (c), and (e) for different Ps. In each case, the delivTime of PSFQ first decreases and then increases as we increase the RMPdelayFactor = δ. When δ is small, the higher priority of RMP in PSFQ does not give enough opportunities for a node x to send Ts and RTs to its neighbors, which can result in more RMPs for some of the neighbors of x. Even if RMPQ(x) = $\varnothing$, the frequent competition for RMP from the neighbors of x decreases x being selected for T or RT, and this too can lead to more RMPs from the neighbors. The result is higher delivTime. On the other hand, when δ is increased beyond a certain value it results in increased idleTime and this increases the delivTime. For a given δ, the increase in P causes fewer RMPs and this decreases the delivTime. Note that the minimum delivTime occurs at a higher δ as P increases.

In OSDRMP, T and RT have priority. So for low δ, the delay of Ts and RTs due to frequent transmission of RMPs is very low and hence initial increase in δ benefits OSDRMP's delivTime much less than that of PSFQ. Like PSFQ, at higher P, increase in δ benefits propagation of successful Ts and RTs through the network and hence, idleTime contributes to increase in delivTime at higher δ in comparison to that at lower P.

For P = 0.3, there are a large number of unsuccessfully transmitted RMPs and hence the delay between RMPs can be effectively utilized by sending more RMPs as in case of OSDRMP(urRMP). But as P increases, more RMPs are successfully transmitted and hence it is good to wait for corresponding RTs instead of sending more RMPs. So the difference in delivTime between OSDRMP with rRMP and with urRMP decreases as P increases, and for P = 0.9 the two versions of OSDRMP have similar delivTime.

4.1.2. Analysis: numMess vs. RMPdelayFactor

For a low δ, highest priority to RMP and frequent transmission of RMPs in PSFQ with delayed Ts and RTs leading to more RMPs, add up to a high numMess for all P. With increase in δ, Ts and RTs have enough time to reach nodes thus reducing the number of RMPs and the numMess for PSFQ. For low δ, both versions of OSDRMP perform better than PSFQ as T and RT have higher priority in OSDRMP. For P = 0.3, unsuccessful transmissions coupled with urRMP and nodes requesting packets which none of their neighbors have (due to OS) contribute to the high numMess for OSDRMP(urRMP) compared to that for PSFQ where there is rRMP and if a node requests a packet p_i, $i < m$

(m is the highest packet sequence number in the node's DC), at least one of its neighbors has it. For large δ and small P = 0.3, the rRMP in OSDRMP(rRMP) eliminates any potential disadvantage of OS and thus the numMess becomes almost equal to that of PSFQ. For P = 0.6 and 0.9, OS combined with successful transmissions result in destinations receiving n packets much faster than PSFQ reducing RMPs and RTs and duplicate Ts to the same node and hence, the numMess required for both versions of OSDRMP with increasing δ is almost equal and at times better than that required for PSFQ.

There is a δ at each P for each protocol after which the numMess will not change with increase in δ. The reason is that all Ts and RTs have been transmitted and it is not time for an RMP from any node. Increase in δ does not benefit this situation and only contributes to idleTime. The δ after which numMess becomes constant increases with increase in P for both OSDRMP(rRMP) and PSFQ. This is because there are more successful transmissions at higher P and increase in δ helps in propagation of successful transmissions. In case of OSDRMP(urRMP), urRMP, unsuccessful transmissions and requests from nodes for packets not present at any of their neighbors at lower P(= 0.3) result in a large number of RMPs and the δ after which numMess do not change is large compared to that at higher P. However as P(= 0.6 and 0.9) increases, the behavior of OSDRMP(urRMP) is similar to that of OSDRMP(rRMP).

4.2. Simulation results for s = 0 and TTL = 14

Figs. 7(a)-(f) show the variations of delivTime and numMess with δ for OSDRMP(rRMP and urRMP) and PSFQ for P = 0.3, 0.6 and 0.9. Here, the same conclusions as those in Fig. 6 hold, except that there are more messages in Fig. 7 (because of larger TTL) which also causes the increase in idleTime start at a higher δ.

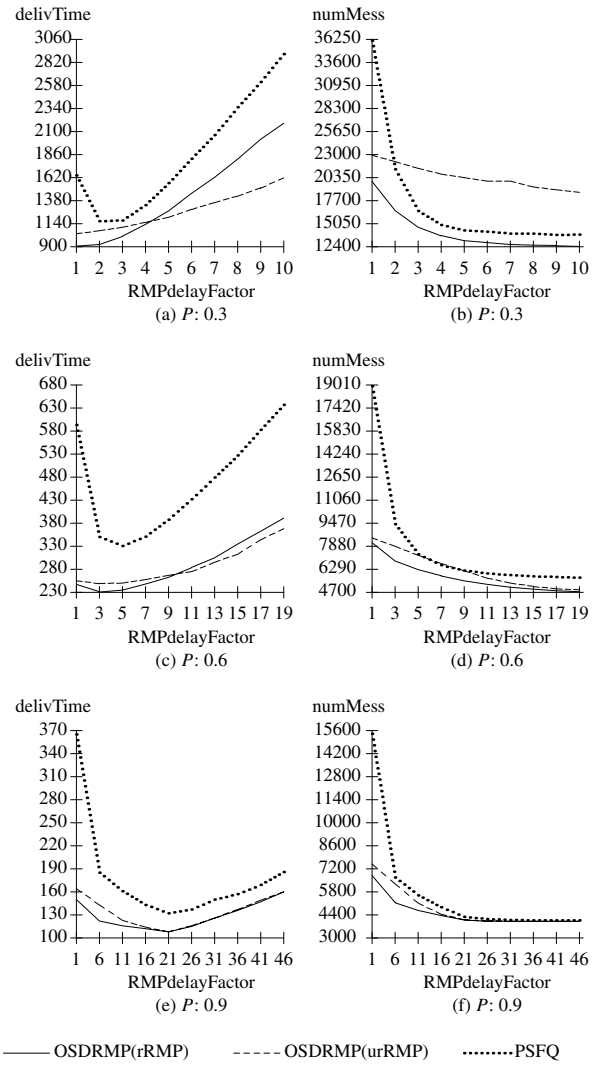


Figure 7. Comparison of OSDRMP and PSFQ for the network in Fig. 5 with s = 0 and TTL = 14.

5. Conclusions

Our simulations demonstrate the superiority of OSDRMP over PSFQ in terms of both the total number of messages and the total delivery time. Indeed, the improvements depend on the network structure and the probability of a successful message transmission. Our future work will consider different priority orders at different nodes based on their nodeTTL (say); it is also possible that nodeTTL can be used more effectively in responding to requests for a packet.

References

[1] C.-Y.Wan, A.T.Campbell and L. Krishnamurthy, "PSFQ: A reliable transport protocol for wireless sensor networks", in Proc. 1st Intl. Workshop on Wireless Sensor Networks and Appl., GA, 2002.

[2] F. Stann and J. Heidemann, "RMST: Reliable data transport in sensor networks", in Proc. 1st IEEE Intl. Workshop on Sensor Network Protocols and Appl., Alaska, May 2003.

[3] Y. Sankarsubramaniam, O. Akan, I. Akyildiz, "ESRT: Event-to-sink reliable transport in wireless sensor networks" in Proc. ACM Mobihoc 2003, Maryland, June 2003.

[4] A. DeSimone, M.C. Chuah and O.-C. Yue, "Throughput performance of transport-layer protocols over wireless LANs", Proc. of IEEE Globecom, Dec 1993.

[5] A. Woo, T. Tong, and D. Culler, "Taming the underlying challenges of reliable multihop routing in sensor networks", ACM Sensys, November 2003.

[6] R. Iyer and L. Kleinrock, "QoS control for sensor networks", IEEE Intl. Conf. on Comm., May 2003.