Adaptive Packet Size Control for Bulk Data Transmission in IPv6 over Networks of Resource Constrained Nodes

Yang Deng, Zhonghong Ou, and Antti Ylä-Jääski

Aalto University, Espoo, Finland
{yang.deng,zhonghong.ou,antti.yla-jaaski}@aalto.fi

Abstract. Conventional transmission in IPv6 over Networks of Resource Constrained Nodes (6lo) favours fixed-size packets and results in low network performance when bulk data transmission is required by applications, for example firmware updating. To tackle this problem, we first investigate performance of bulk data transmission through large packets and make two important observations. Then we propose an adaptive mechanism at IP layer to dynamically adjust packet size in 6lo. We implemented the mechanism on Contiki OS and evaluated it through a series of experiments in Cooja. Experimental results demonstrate that our mechanism outperforms Contiki standard implementation significantly from both reliability and goodput under various network conditions.

Keywords: 6lo, Bulk Data Transmission, Packet Size Control

1 Introduction

IPv6 over Networks of Resource-constrained Nodes (6lo) is a network that provides IPv6 connectivity over constrained nodes such as sensors or actuators. It introduces an adaptation layer to deal with the mapping between IPv6 packets and link frames (hereafter referred to as `packet` and `frame` respectively). Conventionally, when data are transmitted in 6lo, they are firstly encapsulated into a fixed-size packet and then header compression is applied to reduce the packet size. If the compressed packet fits into a single frame then it will be sent out instantly; otherwise fragmentation-reassembly mechanism (hereafter referred to as `6lo fragmentation`) is invoked. The value of the fixed-size is preconfigured empirically (e.g. 140 bytes for IEEE 802.15.4 in Contiki). As RFC4944 [6] points out, in links with a small maximum transmission unit (MTU), the conventional transmission works but with two assumptions: (i) most applications will not use large packets and (ii) application payload is relatively small. Nevertheless, there are application scenarios where these conditions do not hold. Considering firmware updating on-the-fly or massive data exchanging between nodes, both of them involve bulk data transmission for which 6lo conventional transmission is not suitable due to low network performance.

In order to improve 6lo performance of bulk data transmission, we present an adaptive mechanism that adjusts packet size dynamically based on network conditions and utilizes 6lo fragmentation (different from IP fragmentation) to send large packets. Unlike other adaptive mechanisms [2, 3, 7], our mechanism works at IP layer whereas others only work at link layer. Link layer mechanisms highly rely on the link of a specific type. For example the mechanism designed for IEEE 802.15.4 might not be feasible for Bluetooth Low Energy (BLE) or Near Field Communication (NFC), both are documented by 6lo working group. As 6lo operates over links of different types, an upper-layer mechanism is therefore much more desirable. Furthermore, even if frame size can be dynamically controlled by link layer mechanisms, there remains a need for determining IP packet size in 6lo. Unfortunately, to the best of our knowledge, such a mechanism is still missing so far. Our work fills in this gap and in summary it makes three key contributions as follows:

– We investigate performance of bulk data transmission in 6lo through large packets and make two important observations.
– We present design, implementation, and evaluation of IP layer adaptive mechanism suitable for bulk data transmission in 6lo.
– We demonstrate that our adaptive mechanism is able to provide better reliability and higher goodput than Contiki standard implementation under various network conditions.

2 Related Work

Before IPv6 is introduced to Wireless Sensor Network (WSN), frame size optimization for WSN has been studied extensively in literature. Modiano et al. [5] developed a Markov chain model to analyse the channel then performed a maximum likelihood approach to estimate frame size. Sankarasubramaniam et al. [8] used energy-efficiency as the optimization object to determine the best frame size based on a set of radio and channel parameters. The work from [9] discussed the cross-layer solutions to set frame size for different environments like underwater and underground networks. All of these studies focus on finding an optimal fixed size. Another thread of research work suggests using adaptive approaches. Jelenković et al. [3] designed an algorithm to divide the frame into several small chunks to fit available channel periods. Dong's work [2] followed a similar idea that small chunks can be reassembled into a bigger frame. Nonetheless, these solutions work only at link layer. Because 6lo has a wide diversity of links, a solution working at IP layer is beneficial, which is the main contribution of this paper.

3 Mechanism

3.1 Overview

Our motivation comes from the assumption (justified in section 4.2) that bulk data transmission in 6lo by using large packets (invoking 6lo fragmentation if necessary) can improve network performance significantly. Nodes in 6lo usually suffer from intra-path interference [4], which means that when the successor node forwards the packet at the same time it will prevent the reception of following packet coming from the predecessor node. Employing pipeline mechanisms to send packets might mitigate this problem; however, a poorly-designed scheduling policy can cause traffic congestion and lead to an even worse situation. Thus, stop-and-wait Automatic Repeat reQuest (ARQ) is the widely-used protocol to transmit bulk data in 6lo. One good example is Contiki TCP implementation where the window size is 1. In stop-and-wait ARQ, transmission time decreases as packet size increases. Nonetheless, the desire for large packets is limited by network conditions because the whole packet has to be retransmitted if any of the fragments gets lost. To tackle this problem, we propose an adaptive mechanism to adjust packet size based on network conditions; if network condition becomes better the packet size is increased, otherwise it is decreased.

For simplicity and practicality, network conditions are indicated by packet loss rate (PLR) [1]. Through empirical experiments, we make two important **observations**: (1) PLR of large packets is mainly impacted by the number of fragments that the packet is divided into; (2) if frame length is relatively small (approximately 100 octets), it has a trivial influence on the PLR. Due to space limit, we omit the analysis here. Inspired by these two observations, we propose an adaptive mechanism by following two rules:

Rule 1. Adjust packet size by fragments rather than octets, which is deduced from observation (1).

Rule 2. Given the number of fragments, make sure each fragment fills the frame as fully as possible, which is deduced from observation (2).

Based on these two rules, we design two modules, i.e., *Unit Discovery Module* and *Packet Adjustment Module*, to enable the adaptive mechanism.

3.2 Unit Discovery Module

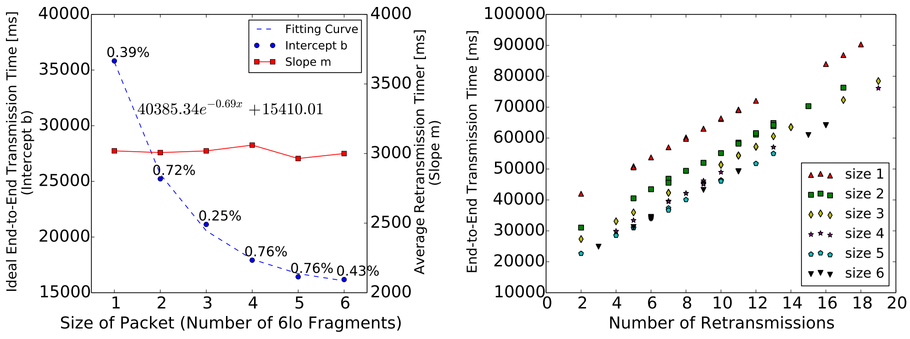
Unit Discovery Module is responsible for finding the unit value by which packet size is increased/decreased in the mechanism. Considering a multi-hop 6lo, the unit value should be the *maximum* size of packet that will not be fragmented by any node along the path from the sender to the receiver. Note that given a packet, the decision on whether it should be fragmented or not is different at different nodes along the path. For example, by default 6lo employs Routing Protocol for Low power and Lossy Networks (RPL) as its routing protocol, the intermediate nodes between the sender and the receiver might insert RPL options into the packet. As a consequence, the packet without need of fragmentation at the sender might be fragmented at an intermediate router. To tackle this problem, we introduce a discovery procedure similar to Path MTU Discovery (PMTUD). Before bulk data transmission starts, the sender pings the receiver using an Internet Control Message Protocol (ICMP) Echo Request message whose size is the current unit value. After an intermediate node receives this message it checks whether 6lo fragmentation is needed or not. If yes, the node drops the message and sends back an ICMP Packet Too Big message containing the new appropriate unit value to the sender. Upon receiving ICMP Packet Too Big message, the sender re-pings the receiver by a new ICMP Echo Request message whose size is the updated unit value. The discovery procedure continues until the sender receives an ICMP Echo Reply message from the target receiver. The unit value corresponding to the receiver is saved in a cache and ready for use in *Packet Adjustment Module*.

3.3 Packet Adjustment Module

Once the unit value is discovered, *Packet Adjustment Module* uses it to adjust packet size according to network conditions, assuming that the routing information is not changed throughout the course of bulk data transmission. Packet size is adjusted by the following equation (note that 6lo fragmentation cuts the packets into 8-octet units):

$$S(n) = \begin{cases} U & n = 1 \\ \lfloor \frac{U - L_{F1}}{8} \rfloor \times 8 + \lfloor \frac{M - L_{FN}}{8} \rfloor \times 8 \times (n-2) + (M - L_{FN}) & n \geq 2 \end{cases} \quad (1)$$

where U is the unit value and M is the link MTU, L_{F1} and L_{FN} (4 and 5 in 6lo) are the length of initial fragment header and non-initial fragment header. It is important to note here that n should be less than 10 as (i) PLR increases significantly as n grows, and (ii) constrained nodes do not have enough memory to collect many fragments.



(a) Negative exponential relationship (b) Linear relationship

Fig. 1: Transmitting bulk data (16KiB) in medium traffic (10s) network with link FER(15%)

Another function of *Packet Adjustment Module* is to adjust packet size efficiently. When running the experiments in section 4.2, we find out that there exists a linear relationship between the end-to-end transmission time and the number of retransmissions, as shown in Fig. 1b. For each packet size we use least squares method to fit the scatters by a linear function $y = mx + b$, and the fitted parameters (slope m and intercept b) with normalized root-mean-square deviation (NRMSD) are plotted in Fig. 1a. Both m and b have practical meaning: m is the average value of retransmission timer (3 seconds in our experiments); whereas b implies the end-to-end transmission time in ideal situation where no retransmission occurs. From Fig. 1a it is clear to see that intercept b against packet size follows a negative exponential function (dashed line in the figure), which means that the performance gain achievable by increasing packet size is significant when packet size is small; as packet size increases, the gain becomes less significant. With this observation, we set up a threshold value (represented as the number of fragments) in our mechanism. When increasing packet size, if the threshold is not reached, we increase it by one fragment every time; and if the threshold is passed, we increase packet size by its current number of fragments, i.e. doubling the size. Conversely, when decreasing packet size, we half the size if the threshold is not reached; and decrease by one fragment if the threshold is passed.

4 Evaluation

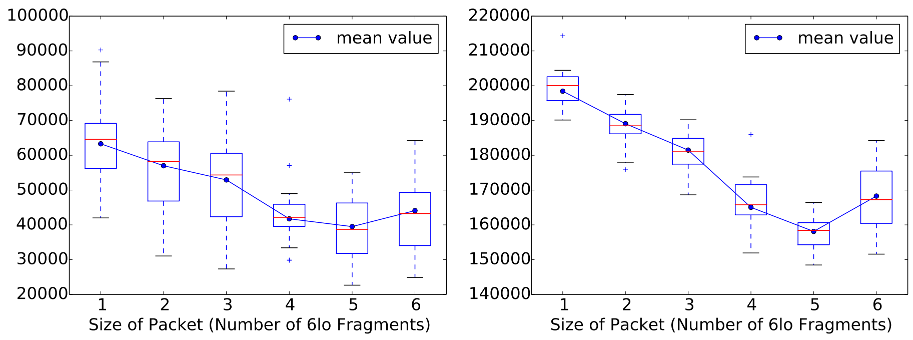
We implemented our mechanism on Contiki OS. For the configuration of network stack in Contiki, we chose 6lowpan as the adaptation layer and employed CSMA at the link layer to provide media access control. As applications usually require bulk data to be transmitted as fast as possible, we used maximum power to transmit the data. To minimize the latency caused by wake-up synchronization among nodes, we switched *contikimac* to *nullrdc*, in which nodes do not sleep. In practice, nodes can switch back to *contikimac* after bulk data transmission completes if necessary.

4.1 Simulation Environment

We simulated a network containing 20 nodes in Cooja, the standard simulator for Contiki. Each node has a transmission range and a larger interference range. Within transmission range, the frame error rate (FER) is able to be configured and it increases as the transmission distance becomes larger; while in interference range, FER is 100% and other nodes are interfered accordingly. We chose one node as the sender (located at one edge of the network) and another node as the receiver (located at the other edge); there were 4 or 5 hops between the sender and the receiver. Moreover, to increase simulation fidelity, we generated background traffic in the network by making each node (except the sender and the receiver) send small packets to random nodes randomly within an interval. If the interval was set to 0, then no background traffic was generated.

4.2 Performance through Large Packets

In section 3.1 we claimed that bulk data transmission in 6lo through large packets can improve network performance significantly. This subsection quantifies this claim. To start with, we consider experiments of medium traffic network (the interval is set to 10 seconds) with different FERs (0%, 15%, 30%, and 45%) where the bulk data (16KiB) are transmitted in different packet sizes. Each experiment was performed 20 times and we use two metrics to compare network performance: end-to-end transmission time, and total transmitted octets by all the nodes along the path from the sender to the receiver. The result from $FER = 15\%$ is shown in Fig. 2. Results from other FER values are similar, thus, are omitted for brevity. The exceptional outputs (represented as outliers) are mainly caused by network congestion resulting from the background traffic. Note that the outliers are excluded when calculating the mean value in the figure. Fig. 2a illustrates that as the packet size increases the end-to-end transmission time decreases accordingly; however, when the packet size exceeds a specific value, the transmission time increases again. The same trend occurs for the total transmitted octets, as shown in Fig. 2b. The reasons for this are explained in section 3.1. From Fig. 2, we can see that if large packets are used, the end-to-end transmission time is improved by 38% (from $\sim$ 65s to $\sim$ 40s), and total transmitted octets are reduced by 20% (from $\sim$ 200KiB to $\sim$ 160KiB).

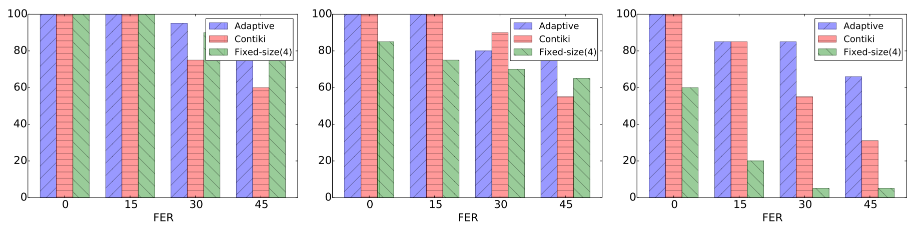


(a) End-to-end transmission time (ms) (b) Total transmitted octets

Fig. 2: Network performance with $FER = 15\%$ for different sizes of packet (represented by number of fragments)

4.3 Mechanism Effectiveness

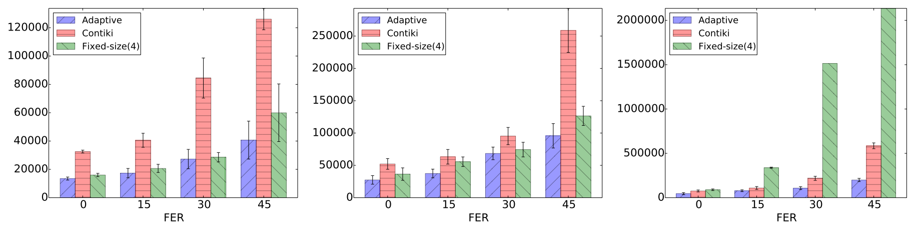
To evaluate the effectiveness of our adaptive mechanism in terms of reliability and goodput, we investigate a series of experiments similar to those in section 4.2 except two more types of background traffic are introduced: low traffic (the interval is set to 0) and high traffic (the interval is set to 5 seconds). Instead of calculating PLR continuously (cf. section 3.1) to determine network conditions, we simply define network conditions as bad (if retransmission occurs) or good (if the payload is successfully acknowledged). The threshold value introduced in section 3.3 is set to 3, which means that the possible number of fragments is 1, 2, 3, or 6. Again, each experiment was performed 20 times.



(a) Low traffic (b) Medium traffic (c) High traffic

Fig. 3: Percentage of successful transmissions [%], the higher the better.

We compare our adaptive mechanism (referred to as Adaptive) with the Contiki standard implementation (referred to as Contiki), as well as the approach using a fixed-size packet (referred to as Fixed-size). Firstly, we focus on the percentage of successful transmissions (i.e. how many tests are completed successfully within the 20 tests), which is an indicator of reliability and robustness. The results are shown in Fig. 3. From the figure, we can see that when network traffic is low and link condition is good, all of the three mechanisms complete 100% of tests. When network traffic increases or link condition becomes worse, the percentage of both Adaptive and Contiki decrease slightly, while Fixed-size drops dramatically. We have discussed the reasons for this behaviour in section 3.1. It is worth mentioning that even in high network traffic with the worst link condition, Adaptive can still complete around 70% of tests, which is significantly higher than that of Contiki. The reason is that the *Unit Discovery Module* in our mechanism ensures that in the worst cases there is no 6lo fragmentation invoked at any node along the path; while in Contiki, intermediate nodes are still possible to trigger 6lo fragmentation that might lead to packet loss in bad network conditions.



(a) Low traffic (b) Medium traffic (c) High traffic

Fig. 4: Estimated end-to-end transmission time [ms], the less the better.

Secondly, we evaluate network goodput using end-to-end transmission time because network goodput is inversely proportional to end-to-end transmission time. Note that if the test gets failed, it is not possible to acquire the output data. Thus, it is unfair to compare only successful results and exclude the failed cases. To make up for this, we introduce a penalty function, which is defined as $1/percentage$. We then compute the estimated results by successful results times penalty function. We present the estimated results in Fig. 4. From the figure, it is clear that regardless of network traffic, overall transmission time for the three mechanisms increases as link conditions get worse. Compared with Contiki and Fixed-size, Adaptive outperforms them in all three network traffic environments. It is also worth noting that despite the decent performance of Fixed-size for low and medium traffic, the transmission time of it for high traffic is extremely high, which demonstrates its severe weakness in such environments. In summary, our adaptive mechanism is able to provide better reliability and higher goodput than the current state-of-the-art approaches in various network conditions.

5 Conclusion

In this paper, we justified the momentum of adjusting packet size adaptively for bulk data transmission in 6lo. Through an empirical study, we made two important observations that inspired the adaptive mechanism design. By evaluating a series of carefully-designed experiments in Cooja, we demonstrated the effectiveness of our mechanism from both reliability and goodput. In the future, we will conduct a systematic study on real devices. Furthermore, power consumption and duty cycles will also be investigated.

References

1. N. Baccour, A. Koubâa, L. Mottola, M.A. Zúñiga, H. Youssef, C.A. Boano, and M. Alves. Radio link quality estimation in wireless sensor networks: A survey. *ACM Transactions on Sensor Networks*, 8(4):34:1–34:33, 2012.
2. W. Dong, X. Liu, C. Chen, Y. He, G. Chen, Y. Liu, and J. Bu. DPLC: Dynamic Packet Length Control in Wireless Sensor Networks. In *Proceedings of IEEE INFOCOM '10*, pages 1–9.
3. P.R. Jelenković and J. Tan. Dynamic Packet Fragmentation for Wireless Channels with Failures. In *Proceedings of ACM MobiHoc '08*, pages 73–82, 2008.
4. 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 ACM SenSys '07*, pages 351–365.
5. E. Modiano. An Adaptive Algorithm for Optimizing the Packet Size Used in Wireless ARQ Protocols. *Wireless Networks*, 5(4):279–286, 1999.
6. G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944 (Proposed Standard), September 2007.
7. Z. Ou, E. Harjula, and M. Ylianttila. Effects of different churn models on the performance of structured peer-to-peer networks. In *Proceedings of IEEE PIMRC '09*, pages 2856–2860, 2009.
8. Y. Sankarasubramaniam, IF. Akyildiz, and S.W. McLaughlin. Energy efficiency based packet size optimization in wireless sensor networks. In *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications*, pages 1–8, 2003.
9. M.C. Vuran and IF. Akyildiz. Cross-Layer Packet Size Optimization for Wireless Terrestrial, Underwater, and Underground Sensor Networks. In *Proceedings of IEEE INFOCOM '08*, pages 780–788, 2008.