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 favours fixed-size packet and results in low network performance when bulk data transmission is required by applications, for example firmware updating. To tackle this problem, we firstly investigate the performance of bulk data transmission through large packets and obtain two important observations. Then we propose an adaptive mechanism to dynamically adjust the packet size based on network conditions. We implement the mechanism on Contiki OS and evaluate it through a series of experiments in Cooja. Experimental results demonstrate that our adaptive mechanism outperforms Contiki standard implementation in terms of reliability and goodput in 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 has an adaptation layer to deal with the mappings between IPv6 packet and link frame (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 it will be sent directly; otherwise fragmentation-reassembly mechanism (hereafter referred to as 6lo fragmentation) is invoked. The value of the fixed-size is preconfigured empirically (e.g. 140 for IEEE 802.15.4 in Contiki). As RFC4944 [6] pointed out, in links with a small maximum transmission unit (MTU) value, this conventional transmission works ordinarily with two assumptions that (i) most applications will not use large packets and (ii) application payload is relatively small. Nevertheless, there are scenarios where these conditions do not hold. Considering the 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 the 6lo performance of bulk data transmission, we present an adaptive mechanism that adjusts the packet size based on network conditions

and utilizes 6lo fragmentation (totally different from IP fragmentation) to send large packets. We implement the mechanism on Contiki OS and evaluate it through a series of experiments in Cooja. Overall, our work makes the following three key contributions:

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

2 Related Work

Before IPv6 is introduced to Wireless Sensor Network (WSN), packet size optimization for WSN has been studied extensively in literature. Modiano et al. develop a Markov chain model to analyse the channel then perform a maximum likelihood approach to estimate the frame size [5]. And Sankarasubramaniam et al. use energy-efficiency as the optimization object to determine the best frame size based on a set of radio and channel parameters [7]. The work from [8] discusses the cross-layer solutions to set the frame size for different environments like underwater and underground networks. All of these studies try to find a fixed size that the network uses. There is another thread of research work that suggests to use adaptive approaches. Jelenković et al. design an algorithm to divide the frame into several small chunks to fit the available channel periods [3]. Dong's work [2] follows a similar idea where small chunks can be reassembled into a bigger frame. Nonetheless, the solutions mentioned above are not suitable for 6lo as they only work at link layer and are limited to a certain type of link. 6lo might run above different types of links and thus 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 fact that bulk data transmission in 6lo by using large packets (invoking 6lo fragmentation if necessary) can improve network performance significantly (as will be shown in section 4.2). 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 next 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 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. Thus, we propose an adaptive mechanism to adjust the 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 obtain 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 design an adaptive mechanism by obeying two rules (listed below) and based on them two modules, namely *Unit Discovery Module* and *Packet Adjustment Module*, are set up.

Rule 1. Adjust the size of packets by fragments rather than octets, deduced from observation 1.

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

3.2 Unit Discovery Module

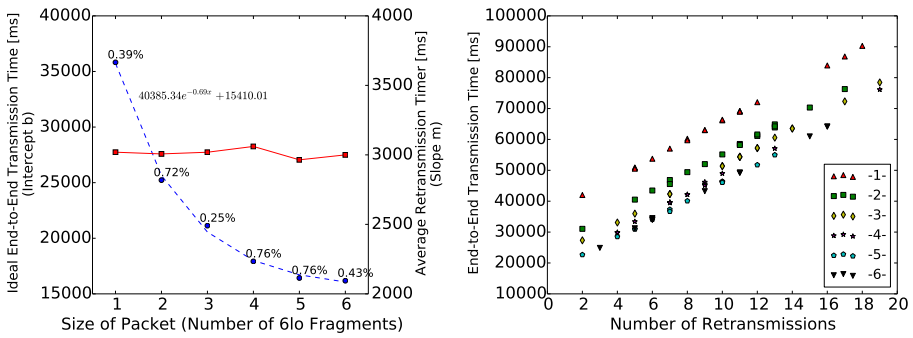
Unit Discovery Module is responsible for finding the unit value by which the size of packet is increased/decreased in the mechanism. Considering a multi-hop 6lo, the unit value should be the *maximum* size of packet that will not be fragmented by any node along the path from the sender to the receiver. 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* starts to use it to adjust the size of packets according to network conditions, assuming that the routing information is not changed throughout the course of bulk data transmission. The size of packets is adjusted by the following equation (noting 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 remark 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 size of packet 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 the size of packet 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 implement our mechanism on Contiki OS and for the configuration of network stack in Contiki, we choose 6lowpan as the adaptation layer and employ CSMA at the link layer to provide media access control. As applications usually require bulk data to be transmitted as soon as possible, we use maximum power to transmit the data. To minimize the latency caused by wake-up synchronization among nodes, we switch *contikimac* to *nullrdc*, in which nodes do not sleep. In practice, nodes can switch back to *contikimac* if necessary after bulk data transmission completes.

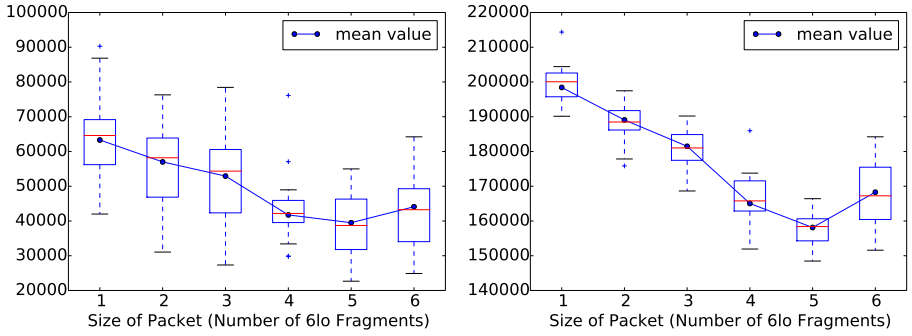
4.1 Simulation Environment

We simulate 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 longer; while in interference range, FER is 100% and other nodes are interfered accordingly. We choose one node as the sender (located at one edge of the network) and another node as the receiver (located at the other edge) and there are 4 or 5 hops between the sender and the receiver. Furthermore, to increase simulation fidelity, we generate 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 is set to 0, then no background traffic is generated.

4.2 Performance through Large Packets

In section 3.1 we argued that bulk data transmission in 6lo through large packets can improve network performance significantly. This subsection quantifies this claim. To start with, we establish a medium traffic network (interval is set to 10 seconds) with different FERs (0%, 15%, 30%, and 45%), then transmit the bulk data (16KiB) in different packet sizes. 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. We run the tests 20 times for each size of packet. 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



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

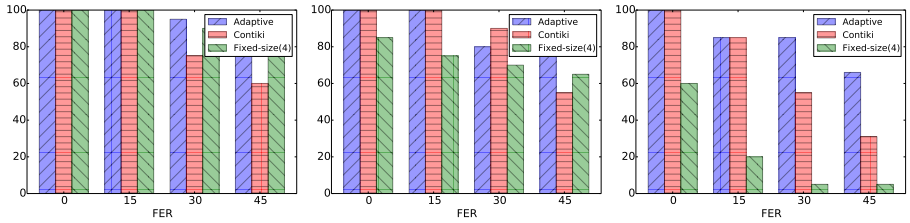
Fig. 2: Network performance with $FER = 15\%$ for different sizes of packet (represented by number of fragments)

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 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).

4.3 Mechanism Effectiveness

To evaluate the effectiveness of our adaptive mechanism in terms of reliability and goodput, we run a series of experiments in similar simulation environments established in section 4.2 but introducing two more types of background traffic: low traffic (interval is set to 0) and high traffic (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. Bulk data 16KiB is used again and every test runs 20 times.

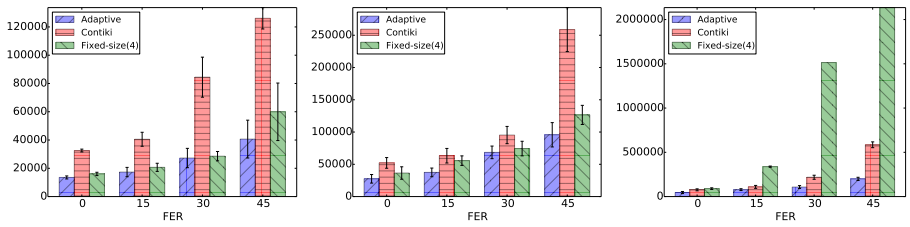
We compare our adaptive mechanism (referred to as Adaptive) with the Contiki standard implementation (referred to as Contiki), as well as the approach using a fixed-size packet (referred to as Fixed-size). Firstly, we focus on the percentage of successful transmissions (i.e. how many tests are completed successfully within the 20 tests), which is an indicator of reliability and robustness.



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

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

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 percentage dramatically. We have discussed the reasons for this behaviour in section 3.1. It is worth mentioning that even in the 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 adaptively adjusting the packet size for bulk data transmission in 6lo. Through an empirical study, we obtained two important observations that inspired the adaptive mechanism directly. By evaluating a series of carefully-designed experiments in Cooja, we demonstrated the effectiveness of our mechanism in term of 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. Nouha Baccour, Anis Koubâa, Luca Mottola, Marco Antonio Zúñiga, Habib Youssef, Carlo Alberto Boano, and Mário Alves. Radio link quality estimation in wireless sensor networks: A survey. *ACM Trans. Sen. Netw.*, 8(4):34:1–34:33, September 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 *INFOCOM, 2010 Proceedings IEEE*, pages 1–9, March 2010.
3. P.R. Jelenković and J. Tan. Dynamic Packet Fragmentation for Wireless Channels with Failures. In *Proceedings of the 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '08, pages 73–82, New York, NY, USA, 2008. ACM.
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 the 5th International Conference on Embedded Networked Sensor Systems*, SenSys '07, pages 351–365, New York, NY, USA, 2007. ACM.
5. E. Modiano. An Adaptive Algorithm for Optimizing the Packet Size Used in Wireless ARQ Protocols. *Wireless Networks*, 5(4):279–286, Jul 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. Updated by RFCs 6282, 6775.
7. Y. Sankarasubramaniam, IF. Akyildiz, and S.W. McLaughlin. Energy efficiency based packet size optimization in wireless sensor networks. In *Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on*, pages 1–8, 2003.
8. M.C. Vuran and IF. Akyildiz. Cross-Layer Packet Size Optimization for Wireless Terrestrial, Underwater, and Underground Sensor Networks. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 780–788, April 2008.