2009 11th IEEE International Conference on High Performance Computing and Communications

6LoWPAN-SNMP: Simple Network Management Protocol for 6LoWPAN

Haksoo Choi, Nakyoung Kim, Hojung Cha
Department of Computer Science
Yonsei University, Seoul, Korea
{haksoo, nykim, hjcha}@cs.yonsei.ac.kr

Abstract

We propose a 6LoWPAN-SNMP that enables transmission of SNMP messages over IPv6-enabled Low-Power Wireless Personal Area Networks (6LoWPAN). The 6LoWPAN-SNMP is an extended modification of the Simple Network Management Protocol (SNMP). Compared to traditional IP-networks, 6LoWPAN is a severely resource-constrained network; hence, existing SNMP protocols need to be modified to meet the goals in RFC 4919, "6LoWPAN Problems and Goals". The proposed 6LoWPAN-SNMP provides for native communication of SNMP messages on LoWPAN networks. The proposed mechanism is resource-efficient and fully compatible with the standard SNMP. It utilizes SNMP header compression and provides extended protocol operations to reduce the number of SNMP messages generated among the SNMP entities. Compatibility with the standard SNMP is achieved by a proxy forwarder on the 6LoWPAN gateway. The proposed mechanism is implemented on actual hardware platforms using the open-source Net-SNMP library and the Berkeley 6LoWPAN on the TinyOS 2.1. The feasibility and effectiveness of 6LoWPAN-SNMP is evaluated. The experimental results show that 6LoWPAN can be effectively supported with network management functionality based on SNMP.

1. Introduction

Low-power Wireless Personal Area Networks (LoWPANs) consist of large numbers of low-cost devices that conform to the IEEE 802.15.4 MAC/PHY layer specification [1]. Due to the low-power and low-cost nature of these technologies, LoWPANs have recently enabled the development of many interesting applications found in industrial, agricultural, vehicular, residential domains, in medical sensors, and in actuator application domains. Compared with similar wireless networks, LoWPAN faces several challenging issues such as small packet sizes, low bandwidth, low power, a large number of devices, unreliability due to uncertain radio connectivity, battery drain, device lockups, and physical tempering [2]. LoWPAN should therefore be equipped with a well-engineered network stack that is able to satisfy the constraints imposed by the resource-constrained nature of LoWPAN. The Internet Protocol (IP), which is the core technical background to the Internet, is a starting point for 6LoWPANs.

The IETF 6LoWPAN working group [3] has developed a method to transmit IPv6 packets over IEEE 802.15.4 networks [4]. A key point of LoWPAN with IPv6 capabilities is to reuse existing IP-based protocols as much as possible. Since LoWPAN is a severely resource-constrained network, management functionality is critical for successful operations. In addition, minimal configuration and self-healing features should also be supported [2]. IP is natively supported by 6LoWPAN nodes that can communicate using standard IP-based protocols such as ICMP and UDP. Moreover, SNMP is widely used in IP networks. The resulting question from the above information is "Why don't we just use SNMP to manage 6LoWPAN?"

Recent research efforts have implemented SNMP on 6LoWPAN or LoWPAN. However, they typically lack the native support of SNMP messages on the LoWPAN nodes because SNMP is usually implemented as a proxy service on a gateway. This proxy-based implementation has a number of issues outlined below.

- **Out-of-date information**: Management information is not retrieved at the time of request. In general, a proxy-SNMP gateway periodically collects management information and stores it temporarily. The information is later retrieved upon request. There is a time-gap between a node's actual status and the stored management information. Even though the retrieved information indicates a normal node status, the node might actually no longer be in that same state.
- **Duplicated implementation of protocols**: To support various SNMP protocol operations, a proxy-SNMP is required to re-implement similar operations on LoWPAN nodes. With the native support of SNMP messages on LoWPAN, the implementation of such a protocol would not be necessary.
- **Gateway overhead**: The proxy-SNMP gateway stores the management information of every node in its network. Protocol conversion should also be implemented to provide compatibility with the standard SNMP. These would also be unnecessary in the case of native support of SNMP.
- **Single point of failure**: If management information is stored in a single gateway, failure of that gateway for a large-scale LoWPAN network connected to multiple gateways would cause all the management information to be unavailable. However, storing the information in every gateway is not practical because each node would then have to send their information to multiple

978-0-7695-3738-2/09 $25.00 © 2009 IEEE
DOI 10.1109/HPCC.2009.49

gateways. A simple yet robust solution would be to natively support SNMP on LoWPAN nodes.

In addition, the standard SNMP protocol should be optimized because a significant traffic overhead is generated by SNMP. In this paper, we solve the aforementioned problems by proposing several techniques to transmit SNMP messages efficiently over a 6LoWPAN. As a result, the 6LoWPAN nodes have the ability to communicate with Network Management Systems (NMS) using standard SNMP protocols.

2. Related Work

The recent release of the RFC standard on 6LoWPAN packet formats [4] has inspired several implementations of IPv6 on LoWPAN. TinyOS 2.1 provides the Berkeley 6LoWPAN implementation [5]. Sensinode Inc. released an open-source version of Nanostack which is a 6LoWPAN implementation on FreeRTOS. Arch Rock, Inc. released PhyNet [6, 7], a wireless sensor network solution based on the 6LoWPAN. Although there are many on-going implementations of the 6LoWPAN, none of them provide for a fully-functional and non-proxy SNMP.

Network management [8] is a critical functionality for 6LoWPANs [2]. The network management system (NMS) should meet the resource-constraints, minimal configuration, and self-healing requirements of LoWPAN. The SNMP architecture [9] and its widely-used NMS applications are ideal for the 6LoWPAN management. The flexible modular architecture of SNMP frameworks is suitable for resource-constrained sensor nodes, because only a desired set of features can be implemented independently and the framework is well-suited for extending the functionalities to support the 6LoWPAN. SNMP requires minimal configuration due to its simple yet powerful protocol and concise operations. The 6LoWPAN can be self-healed by the NMS via SNMP, because the protocol supports the collection of management information as well as the modification of the network configuration in each node. Any problematic situation can therefore be resolved by proper reconfiguration.

A number of commercial or open-source NMS solutions have already been proven to work and have been deployed to manage large-scale networks. Industry leaders such as Cisco manufacture network devices that support SNMP with a standard Management Information Base (MIB) along with their enterprise-specific MIB. Many companies with large networks manage their networks with an SNMP-based NMS. The 6LoWPAN can be easily integrated with existing IP-network management systems by transmitting the SNMP packets over 6LoWPANs.

Recently, several research and industry projects have focused on SNMP implementation over LoWPAN. For instance, a commercial product from Arch Rock supports SNMP as a proxy service on the gateway server. In terms of research, Y. Y. Lim et al [10] proposed a proxy-SNMP for wireless sensor networks. Their implementation is a proxy-based service on a host PC and lacks IP capability on the LoWPAN.

In addition to SNMP header compression, compressing payload should also be considered for efficient implementation. Previous work in this area includes work done by standard organizations such as IETF and IRTF which features ObjectID Delta Compression (ODC) [11] and ObjectID Prefix Compression (OPC) [12], respectively. Although the both Internet drafts are still works in progress, the idea of reducing the size of OID (Object Identifier) information in the SNMP packet is important to the 6LoWPAN-SNMP, because the space occupied by an OID in a SNMP payload is relatively large compared to the actual data. On the other hand, methods to reduce the number of messages generated between agents and managers are proposed in [13]. The key idea is to aggregate several requests and responses whenever possible. In our scheme, we propose techniques that are well-suited for a low-power wireless medium and are able to further reduce the number of messages.

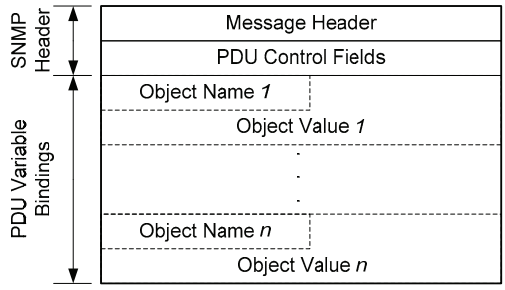

Figure 1. General SNMP message structure.

3. 6LoWPAN-SNMP

6LoWPAN-SNMP consists of several techniques to reduce the amount of total SNMP traffic on the 6LoWPAN network. First, we propose methods to compress SNMPv1 and v2 messages without any modification of the existing protocol operations. These methods mainly reduce the size of each SNMP message. Second, to reduce the number of messages transmitted over the network, we propose new protocol operations and broadcast/multicast support in the SNMP engines. Combining these two techniques, we can reduce large amounts of network traffic while ensuring the correct SNMP operation takes place. Third, the 6LoWPAN-SNMP proxy forwarder is proposed to achieve compatibility with current SNMP versions and to maximize the efficiency of the 6LoWPAN-SNMP. Details of the techniques are explained in the following sections.

3.1. SNMP Header Compression

SNMP header includes every field except the PDU Variable Bindings. It includes the Message Header and the PDU Control Fields as shown in Figure 1. The methods to reduce the size of these fields are now explained. Since each of SNMP versions has different Message Header definitions and PDU Control Fields, each method is specific to each SNMP version. Accordingly, they are separated and explained in different sections. As mentioned in Section 2, ObjectID Delta Compression (ODC) [11] or ObjectID Prefix Compression [12] can be used to compress the Object Name in the PDU Variable Bindings.

Although there are several different message formats for SNMPv2, such as SNMPv2p (Party-based security

TABLE 1. Version Numbers for 6LoWPAN-SNMP

	Version Numbers	SNMP Versions
Current SNMP versions	0	SNMPv1
	1	SNMPv2c
	2	SNMPv2p, v2u
	3	SNMPv3
6LoWPAN -SNMP	4	Compressed SNMPv1
	5	Compressed SNMPv2c
	6	Reserved
	7	Compressed SNMPv3

model), SNMPv2c (Community-based security model), and SNMPv2u (User-based security model), we discuss only SNMPv2c in this paper. This is because all the RFC standards that define the three different security models [14, 15, 16] have become obsolete since the release of SNMPv3. Nevertheless, SNMPv2c is still commonly used in many real-world applications. The RFC that defines SNMPv1 [17] is also obsolete but is also still commonly used. This is reflected by SNMPv3 architecture [9] that defines message processing models for SNMPv1, SNMPv2c, and SNMPv3 in the message processing subsystem.

3.1.1. Compressed SNMPv1 Header. SNMPv1 header consists of Version Number and Community String fields. According to SNMPv1 standard [17], the syntax for the Version Number field is an INTEGER of 4 bytes and the syntax for the Community String is an OCTET STRING of variable size. It is obvious that we do not need the whole 4-bytes number space for the Version Number field because only numbers from 0 to 3 are currently used and we need to double it due to the adoption of the 6LoWPAN-SNMP. Table 1 shows the Version Numbers and the numbers needed by the 6LoWPAN-SNMP. Only numbers from 0 to 7 are required and 3 bits are enough for this number space. While this limits the future extensibility for new protocol version numbers, it is still a reasonable choice especially for networks like 6LoWPAN with a very small MTU size.

The syntax for the Community String is an OCTET STRING of variable size. Although the default community strings are "public" and "private", it is strongly recommended for SNMP administrators to change the default values for security reasons. Therefore, depending on a SNMP administrator's choice, community strings can be of arbitrary length. However, transmitting long community strings in 6LowPAN packets is inefficient because the MTU size is only 127 bytes. In order to handle this problem, we have focused on the fact that the 6LoWPAN nodes are always accessed through the 6LoWPAN gateway. The 6LoWPAN-SNMP proxy forwarder in the gateway can filter out SNMP packets with incorrect community strings and forward SNMP packets with correct community strings without sending the community string field. This scheme makes the end-nodes accept any SNMP packet regardless of the correctness of the community string while still achieving smaller packet sizes. Therefore, in order to increase security, one should enable IEEE802.15.4 MAC layer AES encryption or SNMPv3 should be used.

SNMPv1 standard defines five different PDU types: GetRequest, GetNextRequest, GetResponse, SetRequest, and Trap. They all have common PDU Control Fields except for Trap-PDU. The PDU Type field contains an INTEGER of 4 bytes. Because there are only five PDU types, the size of this field can be reduced in 3 bits. The Request ID field is also an INTEGER of size 4 bytes and contains a number used to match SNMP request and response messages between SNMP entities. Another purpose of the Request ID is to detect duplicated messages in cases where an unreliable datagram service is used [17]. For this purpose, using a 4-byte number space is too much and we can reasonably reduce the number space in 1 byte (0-255). In our experiments with actual 6LoWPAN-SNMP implementation, 1 byte of a request ID successfully detected duplicated messages.

Both Error Status and Error Index fields are an INTEGER of 4 bytes. SNMPv1 standard defines only six error types, so 3 bits are enough for the Error Status field. The Error Index contains a number that indicates which object in the PDU Variable Bindings generated this error. We can reasonably reduce the size of the Error Index in 1 byte because the practical SNMP traffic pattern shows, according to the experimental result from [18], that the size of a response message does not exceed 1400 bytes at most. This indicates less than 140 variable bindings, assuming each variable binding is 10 bytes. Ten bytes are even smaller than the actual Object Name and Object Value bindings in practical applications. Therefore, it is reasonable to reduce the size of the Error Index field to 1 byte. These two fields always contain null value in the request messages, so we just remove the Error Status and the Error Index fields in request messages.

The Trap-PDU Control Field consists of Enterprise, Agent Address, Generic Trap Code, Specific Trap Code, and Timestamp field. The Enterprise field is essentially an OID that indicates the type of object that generated this trap. Because this field is an OID, its size is variable and can be multiple bytes. It is possible to reduce the size of this filed to 1-byte number by maintaining an enterprise OID conversion table on the proxy forwarder in the 6LoWPAN gateway. It is highly unlikely for the gateway to manage nodes from more than 255 different vendors, so 1 byte is a reasonable size. The Agent Address field contains the IP address of the SNMP entity that generated the trap. Obviously, the source IP address already resides in the lower layer headers, so it can be completely removed.

SNMPv1 defines seven different types of generic trap. Therefore, the size of the Generic Trap Code can be reduced in 3 bits, although the syntax for the field is INTEGER of 4 bytes. The syntax of the Specific Trap Code is also an INTEGER and the field contains an enterprise-specific trap code. We can reduce the size of the field in 1 byte if we limit the maximum number of specific trap codes to 255. This limitation is reasonable because most MIB defining specific traps have less than 255 traps in their practical applications. The Time Stamp field has a syntax of TimeTicks of 4 bytes. We can completely remove this field from the SNMP packets by estimating the timestamp at each node. First, the proxy forwarder in a

6LoWPAN gateway maintains a table for recent round-trip time estimation to each node in its network. When the gateway receives a trap message without a timestamp field, it estimates the timestamp of the sending node by subtracting half of the round-trip time from the current timestamp at the gateway. Finally, the gateway adds the timestamp field with its estimated value and forwards the packet. Although the timestamp is an estimation allowing for some errors, this trade off in timestamp accuracy for reduced header size is more favorable in the MTU-size limited 6LoWPAN environment.

3.1.2. Compressed SNMPv2c Header. SNMPv2c standard includes the same community-based security model as SNMPv1. Therefore, it has exactly the same Message Header format as SNMPv1. To compress the SNMPv2c Message Header, we can use the Message Header compression strategy for the SNMPv1 explained in the previous section. The intention of this new SNMP version was to adopt the new protocol operations of SNMPv2 such as GetBulkRequest-, InformRequest-, SNMPv2-Trap-, Report-PDU and more detailed error status fields using the simple SNMPv1 security model [19]. Therefore, we only need to consider those PDU Control Fields for header compression.

The differences in PDU Control Fields between SNMPv2c and SNMPv1 are the PDU Type and the Error Status field. Because RFC3416 [19] introduces new protocol operations, there are nine PDU Types in SNMPv2c (including obsolete value 4). Therefore, only 4 bits are sufficient for representing these types. For the Error Status field, SNMPv2c standard defines 19 different errors, which can be successfully represented by 5 bits. Other than these two fields, we can apply the same methods discussed in the previous section.

RFC3416 [19] defines new PDU Control Fields for the GetBulkRequest-PDU type. This PDU type has two fields different from the other PDU types: Non Repeaters and Max Repetitions. The syntax of the two fields is an INTEGER of 4 bytes. They determine the number of variable bindings in response messages and are especially useful for table traversal operations. In practice, it is rare for the two fields to have a value greater than 255, because doing so may cause fragmentation, which is generally considered to be harmful [20]. For example, assuming each Object Name and Object Value binding has a size of 10 bytes, a value 250 for the Non Repeaters and 0 for the Max Repetitions would generate 2500 bytes for the Response-PDU. Moreover, a value 250 for the Max Repetitions and 0 for the Non Repeaters would generate a much greater size of Response-PDU. A value 250 for both Non Repeaters and Max Repetitions would make it much worse. Considering the default MTU size of 1500 bytes for the Ethernet, such a big packet would be fragmented. In the 6LoWPAN, the MTU size is as small as 127 bytes, so the maximum value of 255 for the Non Repeater and the Max Repetition fields is sufficient. Therefore, these two fields have 1 byte for each in 6LoWPAN-SNMP messages.

3.1.3. Compressed PDU Variable Bindings. PDU variable bindings consist of a series of Object Names and Object Value bindings. As mentioned earlier, the Object Name (OID) field can be compressed by algorithms

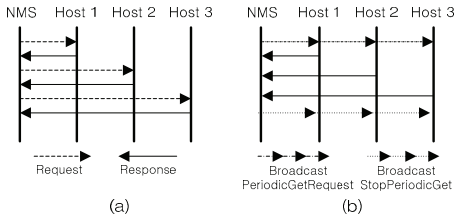

Figure 2. Communication pattern of original SNMP (a), and PeriodicGetRequest / StopPeriodicGet (b)

introduced in [11, 12]. One example of OID compression described in [11] shows the original variable bindings with a total size of 104 bytes become 53 bytes. Another example shows 134 bytes compressed into 87 bytes. Due to the nature of the compression algorithms, the compression ratio becomes more efficient if we request similar OIDs in the GetRequest message.

We can further reduce the size of the PDU Variable Bindings by eliminating the Object Name fields in the response messages and the Object Value fields in the request messages. This is possible because each SNMP entity is able to match incoming response messages to outstanding request messages by using the Request ID field in the PDU Control Fields. As long as the order of the variable bindings is preserved during the communication, each SNMP entity is able to identify the Object Name and the Object Value bindings, even if they are separated in the request and the response messages.

3.2. Extensions to Protocol Operation

The most frequently used communication pattern between SNMP entities is periodically sending request messages and receiving corresponding response messages (e.g. every five minutes). Figure 2 (a) shows such communication behavior. It is clearly beneficial that protocols for resource-constrained networks such as the 6LoWPAN should be designed to minimize the number of message exchanges so as to reduce energy consumption and network congestion. The purpose of the techniques proposed in this section is thus to reduce the number of radio packets transmitted over the network. These new protocol operations are simple, but they can significantly reduce the amount packets transmitted between SNMP entities.

3.2.1. Periodic Get Request / Stop Periodic Get. Instead of sending request messages every time, a 6LoWPAN-SNMP gateway can initiate multiple periodic responses by transmitting a single Periodic Get Request message. If the periodic requests stop, the gateway can transmit a Stop Periodic Get message. Figure 3 shows the message format for Periodic Get Request and Stop Periodic Get messages. The Time Interval field contains the specific time interval in which a requesting SNMP entity wants to receive response messages. This field is 1 byte size with a unit of a minute. Figure 2 (b) shows the communication pattern with the new protocol operations. Table 2 lists the values for the PDU Type field and corresponding protocol operations including the new protocol operations. By using

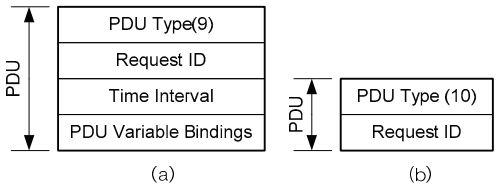

Figure 3. Message body formats for PeriodicGetRequest (a) and StopPeriodicGet messages (b).

TABLE 2. PDU Type Field Definition

PDU Type Values	PDU Type	PDU Type Values	PDU Type
0	GetRequest	6	InformRequest
1	GetNextRequest	7	Trapv2
2	Response	8	Report
3	SetRequest	9	PeriodicGetRequest
4	Not used	10	StopPeriodicGet
5	GetBulkRequest		

these two simple new protocol operations, we can significantly reduce the number of total messages transmitted over a low-power wireless network as it is shown in Section 5.

3.2.2. Broadcast/Multicast SNMP Messages. As shown in Figure 2 (a), network management systems (NMS) using current SNMP versions collect management information mainly by polling each host in the current network. This is because the current SNMP standard is designed only for point-to-point communication [21]. By adopting the broadcasting and multicasting of SNMP messages, we can reduce the overhead of polling individual nodes in the network and exploit the broadcast nature of the wireless medium. Instead of polling every host in the network periodically, a NMS can inject a broadcast Get Request message into the network. Every host will then immediately respond with a Response message to the NMS. When combined with the Periodic Get Request message described in the previous section, a single message can initiate multiple periodic Responses from every host in the network.

Although the benefits of broadcast and multicast messages are significant, the actual implementation seems to be simple. We have performed preliminary experiments using tethereal, a widely used packet monitoring tool, and Net-SNMP, an open source package of SNMP tools. The current version of the command responder in the Net-SNMP responds to the broadcast and multicast request messages. However, the snmpget application, which transmits the Get Request, receives only one response message and then self-terminates even though multiple response messages have arrived at the network interface. Therefore, only small changes to the snmpget application are needed to enable the receiving of multiple response messages initiated by the broadcast or multicast request messages.

3.3. Proxy Forwarder on 6LoWPAN Gateway

As defined in [22], the operation of the SNMP proxy forwarder is forwarding SNMP requests to other SNMP entities irrespective of the specific managed objects. One of the main features of the proxy forwarder is to convert SNMP messages from one version to another. For example, when a SNMP entity using SNMPv3 protocol wants to communicate with another SNMP entity using SNMPv1, a proxy forwarder can mediate the communication by converting SNMPv3 messages into SNMPv1 messages and vice versa. The main purpose of the proxy forwarder on the 6LoWPAN gateway is exactly the same. To achieve maximum interoperability between current versions of SNMP and the 6LoWPAN-SNMP, the proxy forwarder translates the messages of current SNMP versions into corresponding 6LoWPAN-SNMP messages and vice versa, applying the header compression techniques described in the previous sections at the same time.

Moreover, in order to support the new protocol operations discussed in this paper, the 6LoWPAN-SNMP proxy forwarder should be able to automatically convert the original protocol operations into the new ones. This conversion feature includes the automatic detection of periodic Get Requests and of a SNMP broadcast/multicast. Because network management systems based on current SNMP versions will still send Get Request messages to periodically poll every host in the network, the 6LoWPAN-SNMP proxy forwarder should have the ability to efficiently optimize the request messages to reduce the size and number of messages transmitted over the 6LoWPAN network. This feature operates as follows. First, the proxy forwarder operates in normal mode by performing SNMP message conversion and forwarding. When the proxy forwarder detects the same multiple Get Request messages coming in regularly at a specific time interval, it stops forwarding the multiple Get Request messages. Instead, it generates a Periodic Get Request message and transmits it once, initiating periodic responses from the receiving SNMP entities. The proxy forwarder keeps monitoring the incoming Get Request messages, but does not forward it. Once the proxy forwarder detects any changes in the content of the incoming Get Requests, such as changes in time interval, a change in the managed objects, or no more incoming Get Requests, it retransmits a Periodic Get Request with a new time interval, managed object, or Stop Periodic Get message, respectively. By converting the multiple Get Requests polling on each host to a single broadcast or multicast, Get Request can be performed in similar fashion. Combining these two automatic protocol conversions can greatly reduce the amount traffic on the low-power wireless network.

4. Implementation

4.1. 6LoWPAN-SNMP Agent

We implemented the 6LoWPAN-SNMP agent using the nesC programming language [23] with TinyOS 2.1. The agent is developed on top of b6loWPAN [5], the 6LoWPAN implementation from the TinyOS community. The 6LoWPAN-SNMP agent consists of four components: a message dispatcher, a message processor, a command

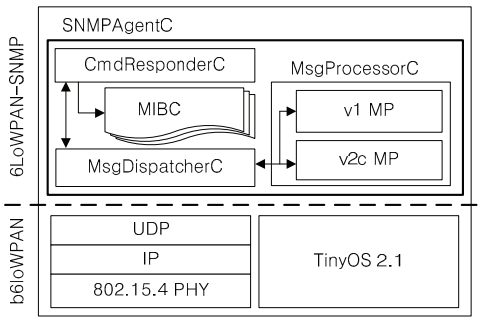

Figure 4. 6LoWPAN-SNMP mote agent component structure

responder, and an MIB component. Figure 4 shows how they are connected with other components.

The MsgDispatchC, the message dispatcher component implements the sending/receiving of 6LoWPAN-SNMP messages to/from the network. It first determines the version of a received message so that the MsgProcessC can extract data from the message by applying different message processing models depending on the version. Currently two versions, SNMP version 1 and version 2c, are supported by 6LoWPAN-SNMP.

The MIBC component implements the hierarchical structure of 6LoWPAN MIB, i.e. LOWPAN-TC-MIB and IPV6LOWPAN-MIB [24]. Moreover, the MIBC component is also capable of OID completion to support GetNextRequest and GetBulkRequest PDU types. In addition to the 6LoWPAN MIB objects, we have included several private objects such as the number of messages sent, forwarded, dropped, and so on. These objects are used in our experiment to collect information about the network status.

The CmdRespondC is responsible for processing the SNMP PDU and variable bindings. After processing the received SNMP message, the command responder generates an adequate response PDU for the received message. In particular, in the case of a PeriodicGetRequest, the command responder registers a timer that generates a Response message periodically as if a GetRequest message has arrived periodically.

4.2. 6LoWPAN-SNMP Proxy Forwarder

The proxy forwarder on the 6LoWPAN gateway is implemented using the Net-SNMP open source library and the b6loWPAN gateway. One of the roles of the proxy forwarder is to convert the SNMP messages into 6LoWPAN-SNMP compressing headers and variable bindings. While most of the messages are converted and forwarded, the continuous reception of the same GetRequest messages is specially handled. As mentioned in the previous section, if the proxy forwarder periodically receives the same SNMP requests, the next request will be converted into a PeriodicGetRequst message. One of the important considerations in implementation is that the proxy forwarder has to ensure the successful delivery of a PeriodicGetRequst to a destination, or successive requests will keep failing. The default transport layer protocol for SNMP is UDP, but the b6loWPAN does not provide any reliable communication. Therefore, we have implemented a simple retransmission scheme using an ACK message for the PeriodicGetRequest message.

TABLE 3. Header Compression Comparison (bytes)

Header Type	Original	Comp-ressed	Ratio
SNMPv1, v2 Header	10	1	10.0%
SNMPv1, v2 Common PDU	26	4	15.4%
SNMPv1 Trap-PDU	40	4	10.0%
SNMPv2 GetBulkRequest-PDU	26	4	15.4%

TABLE 4. The Size of Compressed PDU Variable Bindings (bytes)

Varbinds Type	Original SNMP	6LoWPAN-SNMP	Ratio
Request Varbinds	100	41	41.0%
Response Varbinds	104	20	19.2%

5. Evaluation

In this section, we provide the results of the header compression techniques, experimental results, and the memory and code footprint of the 6LoWPAN-SNMP mote agent to show the effectiveness and feasibility of the 6LoWPAN-SNMP for transmitting SNMP messages over the low-power wireless 6LoWPAN.

5.1 Compressed Header Size

Table 3 summarizes the results of the header compression techniques of the 6LoWPAN-SNMP. The Community String field is assumed to be “public” and the Enterprise field is assumed to be 10 bytes long. As can be seen from the table, the header compression techniques proposed in this paper achieve an average 12.7% of the original SNMP header size in the case of SNMPv1 and v2. The range of the compressed header size varies from 1 to 4 bytes. Considering the MTU size of the 6LoWPAN, the compressed header size of the 6LoWPAN-SNMP is small enough and gives sufficient space for the PDU Variable Bindings without causing fragmentation.

Table 4 shows the result of the PDU Variable Bindings compression. We have assumed the OID Delta Compression and example variable bindings from [11]. In the case of the Request variable bindings, 41% of the compression ratio is achieved mainly by the OID Delta Compression. A further compression ratio of 19.2% on the Response variable bindings is achieved by the elimination of the Object Name fields in the response message as described in section 3.1.4. Using these variable bindings, the total amount of traffic is analyzed for both the original SNMP and the 6LoWPAN-SNMP.

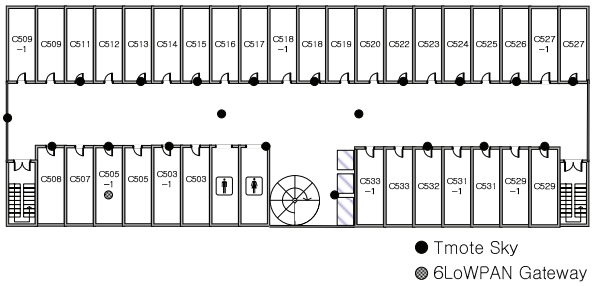

Figure 5. Testbed deployment

TABLE 5. Size of SNMP Messages Used in the Experiments (bytes)

Traffic Type	Original SNMP	6LoWPAN-SNMP	Ratio
GetRequest	46	5	10.9%
Response	112	40	35.7%
PeriodicGet	N/A	5	N/A
StopPeriodic	N/A	2	N/A

5.2. Experimental Results

In order to show that the 6LoWPAN-SNMP successfully reduces the total amount of traffic generated by a network management system and guarantees better network condition, we have performed comparative experiments using both the original SNMP message formats and 6LoWPAN-SNMP header compression with the PeriodicGet/Stop protocol operations. As shown in Figure 5, 21 Tmote Skys are deployed on our testbed covering the whole 5th floor of the 3rd Engineering Building at Yonsei University. With maximum radio power, the testbed network takes 5-6 hops from the base station to the end-nodes. The remote network management system is placed outside the university network, and has 24 routers between the 6LoWPAN Gateway and the remote system. The remote network management system is implemented with the Net-SNMP library and sends a SNMPv2 GetRequest message every five minutes to every node in the LoWPAN network. The network traffic information is collected using the GetRequest operation. The sizes of SNMP messages used in our experiments are summarized in Table 5.

Moreover, to make the experiments as close to real-world scenarios as possible, we have generated a typical workload on the experimental network setup, where every node in the network sends its sensor readings (temperature, humidity, light, and battery voltage) to the remote host every five minutes. The remote host is also placed on the same network with the network management system.

5.2.1. Traffic Changes on Gateway. Figure 6 shows how the network traffic on the 6LoWPAN gateway changes in the case of using both original SNMP and 6LoWPAN-SNMP protocol operations. In this experiment, only the PeriodicGet/Stop protocol operation is used in connection with the 6LoWPAN-SNMP header compression. The graphs show the number of messages that were forwarded

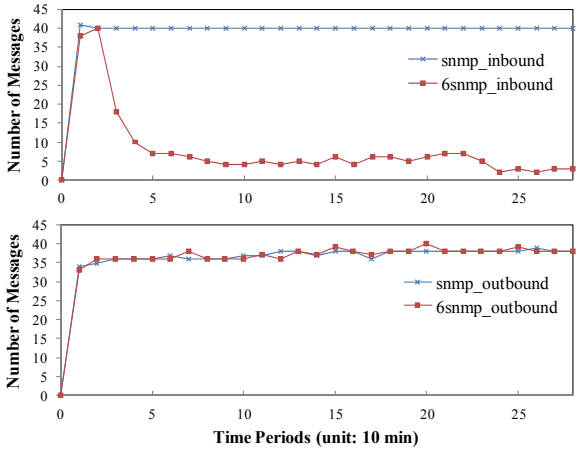

Figure 6. Network traffic on 6LoWPAN gateway

TABLE 6. Average Number of Messages per 10 minutes on Gateway

Traffic Type	Original SNMP	6LoWPAN-SNMP	Ratio
Inbound	38.7	7.7	19.9%
Outbound	35.9	35.9	100.0%

on the gateway in units of 10 minutes. We compare two different types of forwarded messages: inbound (from the network management system to the LoWPAN nodes, that is GetRequest messages) and outbound (from the LoWPAN node to the network management system, that is Response messages). The average number of messages for inbound and outbound traffic are summarized in the Table 6, respectively.

As we can see in Figure 6, the changes in the number of outbound messages for both SNMP and 6LoWPAN-SNMP is almost identical. This is because the 6LoWPAN-SNMP gateway simply forwards the outbound messages as they arrive at the gateway. However, the number of messages for inbound SNMP packets is significantly reduced. In the time period of 1 to 3 (10 to 30 minutes), the number of inbound messages for both the original SNMP and 6LoWPAN-SNMP gateway increases in tandem. However, once the 6LoWPAN-SNMP gateway detects periodicities on inbound GetRequest messages, it converts the request messages to PeriodicGet messages. This behavior is reflected in the rapidly decreased number of messages in the time period of 3 to 6. After sending PeriodicGet messages, the 6LoWPAN-SNMP gateway keeps monitoring the PeriodicGet message to see if it has arrived at the end-nodes and retransmits the PeriodicGet message when no response is heard at the gateway within a pre-determined timeout period. In this experiment, one node did not have good link connectivity with the base station, so every time the gateway received the GetRequest message, it retransmitted the PeriodicGet request two or three times. (The maximum number of retries was four, including initial transmission.). This bad connectivity lasted until the end of the experiment, and we can see this behavior in time period of 6 to 29 in the graphs.

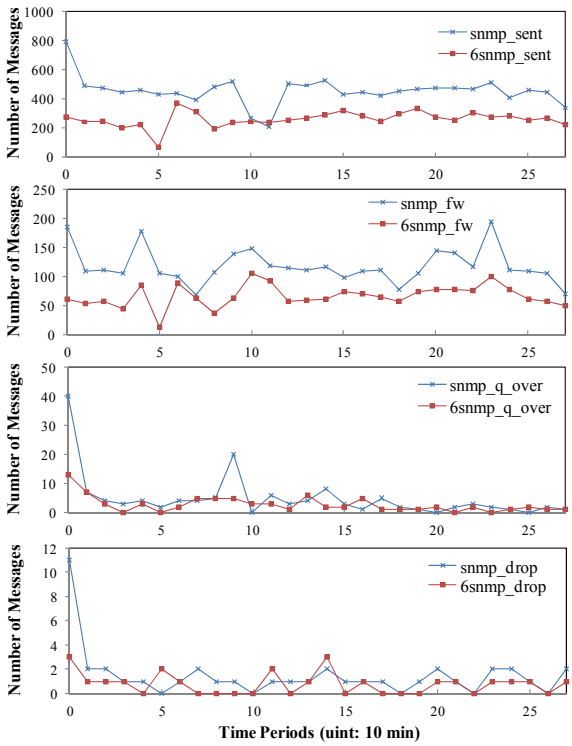

Figure 7. Network traffic in the LoWPAN

TABLE 7. Average Number of Messages per 10 minutes on the Entire LoWPAN Network

Traffic Type	Original SNMP	6LoWPAN-SNMP	Ratio
Sent	450.9	254.8	56.5%
Forwarded	118.4	66.9	56.5%
Dropped due to queue overflow	4.9	2.8	57.14%
Dropped due to bad link	1.5	0.8	53.33%

5.2.2. Traffic Changes in the LoWPAN Network. Figure 7 compares the various metrics for the status of the LoWPAN network when the original SNMP and 6LoWPAN-SNMP messages are used. The graphs show the changes in number of messages in a unit of 10 minutes. The values on the Y-axis are the total number of messages generated by every node in the LoWPAN network. In other words, it shows the number of messages on the network in various metrics at each time period. The metrics used in this experiment are the number of messages that are sent, forwarded, dropped due to queue overflow, and dropped due to bad link connectivity. The average number of messages for the various metrics are summarized in Table 7.

In general, 6LoWPAN-SNMP has approximately half the number of messages that are sent and forwarded in the network compared to the original SNMP. Similarly, the number of messages that are dropped due to either queue overflow or bad link connectivity for the 6LoWPAN-SNMP is also half that of the original SNMP. The results show that the 6LoWPAN-SNMP significantly increases network performance by reducing almost by half the size and number of packets that are transmitted and dropped. The SNMP messages are compressed to fit into one fragment by using 6LoWPAN-SNMP header compression techniques while the original SNMP messages are fragmented due to their big message size of more than 100 bytes. Moreover, converting regularly arriving GetRequest messages into PeriodicGet messages significantly reduces the number of packets transmitted in the network.

In addition, the original SNMP shows a very high number of sent and dropped packets in the first 10 minutes. The 6LoWPAN-SNMP also shows a slightly higher number of sent and dropped messages in the first 10 minutes. Our analysis shows that this is caused by the route update messages of the b6loWPAN stack. When the mote initially boots, it multicasts a router solicitation message and receives a router advertisement. Then, the mote periodically sends route update messages to the base station. The timer interval for the route update message was originally 15 seconds. This interval was too frequent for our experiment, so we adjusted this timer to be fired every 15 seconds only for the first five minutes and then every five minutes. In this way, the nodes could quickly participate in the network and then reduce the number route update messages once the network routes were stabilized. Therefore, at the first stage of the experiment, there were lots of route update messages generated by every node other than the SNMP messages, causing queue overflows and packet drops. It was worse for the original SNMP because there were even more packets due to fragmentation.

TABLE 8. Memory and Code Footprint for Mote Agent (bytes)

Components	ROM	RAM
SNMPAgentC	212	46
MsgDispatchC	476	112
MsgProcessC	6,018	211
CmdRespondC	2,136	212
MIBC	3,588	40
Total	12,430	633
TinyOS + b6loWPAN	21,128	3,939
Total	33,622	4,572

5.3. Memory and Code Footprint

To estimate the overhead of 6LoWPAN-SNMP, we measured the code and memory footprints of each component in the 6LoWPAN-SNMP agent. The 6LoWPAN-SNMP mote agent uses 12,430 bytes of ROM and 633 bytes of RAM. These constitute 25.28% of program ROM and 6.18% of RAM on a Tmote Sky platform. Table 8 shows the detailed breakdown of the code size and memory footprint. The MsgProcessC component occupies almost half of the agent code size because the implemented code for 6LoWPAN-SNMP header parsing and compression reside in this component.

Most of the RAM usage is occupied by buffer memory for message parsing and compression.

6. Conclusion

SNMP should be natively supported by 6LoWPAN because proxy-SNMP has several limitations. No previous research or implementation provides the native communication of SNMP messages over the 6LoWPAN.

The main contribution of this paper is to present techniques to transmit the SNMP packets over the 6LoWPAN, which are compatible with the existing SNMP standard and optimized for the resource-constrained nature of the low-power and low data- rate wireless networks. First, the header compression techniques for SNMPv1 and v2c are proposed and they achieve an average 12.7% of the compression ratio and compressed header size of 1 to 4 bytes. With examples used in [11], the methods to compress the SNMP payload, i.e. PDU Variable Bindings, achieve 41% and 19.2% of the compression ratio for Request and Response messages, respectively. Second, extended protocol operations such as PeriodicGetRequest and broadcast/multicast SNMP messages are proposed. These effectively reduce the number of packets that are transmitted. By using SNMP message compression and the extended protocol operations together, we can further reduce the total amount of network traffic in a LoWPAN network. Third, the 6LoWPAN-SNMP gateway and mote agent for SNMPv1 and v2 are implemented on an actual sensor node platform and deployed on a multihop testbed with 21 nodes and 5-6 hops. Experimental results show that the 6LoWPAN-SNMP successfully reduces the total amount network traffic by approximately half and also reduces the number of packets dropped due to queue overflow or bad link connectivity.

Future work includes analyzing the feasibility of compressing SNMPv3 message headers and implementing authentication and encryption schemes of SNMPv3 on a resource-constrained sensor hardware platform.

7. Acknowledgements

This research was supported by the National Research Laboratory (NRL) program of the Korean Science and Engineering Foundation (No. M10500000059- 6J0000-05910) and the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute for Information Technology Advancement)" (IITA-2009-C1090-0902-0015).

8. References

[1] IEEE Computer Society, “Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs),” IEEE 802.15.4 Specification, 2006.

[2] N. Kushalnagar, G. Motenegro, C. Schumacher, “IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals,” RFC 4919, 2007.

[3] IPv6 over Low power WPAN (6LoWPAN), IETF WG, http://www.ietf.org/html.charters/6lowpan-charter.html

[4] G. Motenegro, N. Kushalnagar, J. Hui, D. Culler, “Transmission of IPv6 Packets over IEEE 802.15.4 Networks,” RFC 4944, 2007.

[5] b6loWPAN, Berkeley 6LoWPAN Implementation

[6] “Arch Rock IP/6LoWPAN Overview: An IPv6 Network Stack for Wireless Sensor Networks,” White Paper, Arch Rock, Inc., http://www.archrock.com/downloads/Arch_Rock_6LoWPAN-Whitepaper.pdf

[7] G. Tolle, “Demo Abstract: A 6LoWPAN Application Environment,” The Fifth International Conference on Embedded Networked Sensor Systems (Sensys), 2007.

[8] D.W. Stevenson, “Network Management: What it is and what it isn't,” White Paper, 1995, http://www.itmweb.com/essay516.htm

[9] D. Harrington, R. Presuhn, B. Wijnen, “An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks,” RFC 3411, 2002.

[10] Y. Y. Lim, M. Messina, F. Kargl, L. Ganguli, M. Fischer, T. Tsang, "SNMP-Proxy for Wireless Sensor Network," Fifth International Conference on Information Technology: New Generations (ITNG), 2008.

[11] J. Schoenwaelder , “SNMP Payload Compression,” draft-irtf-nmrg-snmp-compression-01.txt, Internet-Draft, Work in Progress, 2001.

[12] W. Hardaker, “Object Oriented Protocol Operations for the Simple Network Management Protocol,” draft-ietf-eos-oops-00.txt, Internet-Draft, Work in Progress, 2003.

[13] G. Keeni, “The Managed Object Aggregation MIB,” RFC 4498, 2006.

[14] J. Galvin, K. McCloghrie, “Administrative Model for version 2 of the Simple Network Management Protocol (SNMPv2),” RFC 1445, 1993.

[15] J. Case, K. McCloghrie, M. Rose, S. Waldbusser, “Introduction to Community-based SNMPv2,” RFC 1901, 1996.

[16] G. Waters, “User-based Security Model for SNMPv2,” RFC 1910, 1996.

[17] J. Case, M. Fedor, M. Schoffstall, J. Davin, “A Simple Network Management Protocol (SNMP),” RFC 1157, 1990.

[18] J. Schonwalder, A. Pras, M. Harvan, J. Schippers, R. Meent, “SNMP Traffic Analysis: Approaches, Tools, and First Result,” 10th IFIP/IEEE International Symposium on Integrated Network Management, 2007.

[19] R. Presuhn, J. Case, K. McCloghrie, M. Rose, S. Waldbusser, “Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP),” RFC3416, 2002.

[20] C. Kent, J. Mogul, "Fragmentation Considered Harmful," In the Proceedings of ACM SIGCOMM '87, Stowe, VT, August 1987.

[21] D. Kwak, J. Kim, “Design of Network Management System Employing Secure Multicast SNMP,” in Proceedings of 4th International Conference on Networking (ICN 2005), Reunion Island, France, April 2005

[22] D. Levi, P. Meyer, B. Stewart, “Simple Network Management Protocol (SNMP) Applications,” RFC 3413, December 2002

[23] D. Gay, P. Levis, R. Behren, M. Welsh, E. Brewer, D. Culler, "The nesC language: A holistic approach to networked embedded systems," in Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation, San Diego, CA, 2003

[24] K. Kim, H. Mukhtar, S. Yoo, S. Daniel Park, “6lowpan Management Information Base”, IETF Internet-Draft, work in progress, 2008