System Evaluation and Design of Delay-Sensitive Wireless Networks

A Dissertation Presented

by

Xi Deng

to

The Graduate School

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Computer Engineering

Stony Brook University

September 2013

Stony Brook University

The Graduate School

Xi Deng

We, the dissertation committee for the above candidate for the Doctor of Philosophy degree, hereby recommend acceptance of this dissertation.

Yuanyuan Yang – Dissertation Advisor
Professor, Department of Electrical and Computer Engineering

Sangjin Hong – Chairperson of Defense
Professor, Department of Electrical and Computer Engineering

Milutin Stanacevic
Associate Professor, Department of Electrical and Computer Engineering

Jie Gao
Associate Professor, Department of Computer Science

This dissertation is accepted by the Graduate School.

Charles Taber
Interim Dean of the Graduate School

Abstract of the Dissertation

System Evaluation and Design of Delay-Sensitive Wireless Networks

by

Xi Deng

Doctor of Philosophy

in

Computer Engineering

Stony Brook University

2013

Wireless networks have now been widely used and enjoyed in many aspects of human life and the society. Delay-sensitive wireless network, in a strict sense, is not a particular type of wireless networks. It represents a large number of wireless networks whose performance is greatly affected by the packet end-to-end delay, defined as the duration between a packet's generation at the source node and its delivery to the destination. Such network has been growing rapidly in recently years due to the tremendous demands from network applications. Emerging multimedia applications including Video-on-demand (VOD) and Voice-over-IP (VoIP) have become one of the major traffic types that require a large amount of data packets to be delivered in a timely and continuously fashion; various monitoring applications, such as emergency detection and battlefield surveillance, also call for delay-sensitive networks, where a faster delivery of monitoring data may save tremendous economical or military loss by preventing some critical hazards.

Compared to wired network, achieving low packet delay is a more challenging work in wireless networks because the resources in wireless networks are much more constrained. This dissertation focuses on the study of resource constraints at network nodes. Compared to transmissions, resource constrained network nodes have not been paid full attention to in previous works as their impact may not be significant at earlier wireless networks. With increasing demand of delay-sensitive networks, these limitations now become performance bottlenecks and worthy study. The contributions of this dissertation are two-fold. First, we designed and implemented a general, flexible hardware-aware network platform which takes hardware processing behavior into consideration to accurately evaluate network performance. Similar to the lack of study of network node resources, there also lacks of an experimental and evaluation tool that is suitable for this particular study. With our platform, the nodal processing can now be accurately modeled and evaluated in the form of network simulations. This platform facilitates the remaining part of this dissertation and any other nodal processing related researches for the whole research community as well. Second, we considered practical issues in delay-sensitive wireless sensor networks as most sensor nodes are resource-constrained devices. We studied the impact on packet delay caused by two types of resource constraints: limited hardware processing capability and half-duplex antenna. Algorithms have been designed to minimize the negative impact of these resource constraints to better support delay-sensitive applications.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Wireless networks have now been widely used and enjoyed in many aspects of human life and the society. Delay-sensitive wireless network, in a strict sense, is not a particular type of wireless networks. It represents a large number of wireless networks whose performance is greatly affected by the packet end-to-end delay, defined as the duration between a packet's generation at the source node and its delivery to the destination. Such network has been growing rapidly in recently years due to the tremendous demands from network applications. Emerging multimedia applications including Video-on-demand (VOD) and Voice-over-IP (VoIP) have become one of the major traffic types that require a large amount of data packets to be delivered in a timely and continuously fashion; various monitoring applications, such as emergency detection and battlefield surveillance, also call for delay-sensitive networks, where a faster delivery of monitoring data may save tremendous economical or military loss by preventing some critical hazards.

Compared to wired network, achieving low packet delay is a more challenging work in wireless networks because the resources in wireless networks are much more constrained. Resource constraints reside in both the transmission medium and network nodes. As for transmission, wireless network is known to have lower bandwidth than wired networks. The sharing nature of wireless channel is another limitation as wireless transmission can be interfered by other transmissions or noises, resulting in more protection mechanism and thus longer packet delay. As for network nodes, resources include computation capability, memory, antenna and energy, affecting the packet delay in different ways. Wireless network consists of various types of nodes with some (e.g. sensor nodes) very restrict on these resources. Weak computation capability prolongs the nodal processing of network algorithms and protocols, which is a part of packet delay. Small memory disables the network node to store a considerable amount of information necessary to perform certain network optimization for delay reduction. Half-duplex antenna substantially slows down the transmission procedure. Battery powered devices in wireless networks with limited power supply tend to adopt more energy conservation approaches that may increase the packet delay.

This dissertation focuses on the study of resource constraints at network nodes. Compared to transmissions, resource constrained network nodes have not been paid full attention to in previous works as their impact may not be significant at earlier wireless networks. With increasing demand of delay-sensitive networks, these limitations now become performance bottlenecks and worthy study. The contributions of this dissertation are two-fold. First, we designed and implemented a general, flexible hardware-aware network platform which takes hardware processing behavior into consideration to accurately evaluate network performance. Similar to the lack of study of network node resources, there also lacks of an experimental and evaluation tool that is suitable for this particular study. With our platform, the nodal processing can now be accurately modeled and evaluated in the form of network simulations. This platform facilitates the remaining part of this dissertation and any other nodal processing related researches for the whole research community as well. Second, we considered practical issues in delay-sensitive wireless sensor networks as most sensor nodes are resource-constrained devices. We studied the impact on packet delay caused by two types of resource constraints: limited hardware processing capability and half-duplex antenna. Algorithms have been designed to minimize the negative impact of these resource constraints to better support delay-sensitive applications.

1.1 Construction of Hardware-aware Network Experimental Platform

Accurate performance evaluation of delay-sensitive networks requires consideration of every factor that contributes to the packet delay. Hardware processing is often neglected in the network evaluation. Many protocols and algorithms are evaluated either regardless of the impact of hardware processing or simply regarding the entire processing as a constant overhead. In addition, when computation resource is limited, algorithms are designed of low complexity to keep processing impact low. In this case, a common assumption is that the processing speed is much faster than the communication speed so that processing will not substantially affect the overall network performance.

Such assumption does not hold with recent research advances. Sophisticated network algorithms and applications, such as header compression, packet encryption, image and video processing, are introduced into network to improve network performance and functionality, greatly increasing the complexity of hardware processing, which has a significant impact on network latency and bandwidth. On the other hand, nodes in wireless network are more computational resource-constrained, and hence more prone to be affected by such complex processing. Thus it is necessary to reconsider hardware processing in network performance evaluation.

Current evaluation tools can be mainly categorized into network simulators and emulators. Popular general network simulators include NS-2 [2], GlomoSim [3] and OPNET [1]. While they are powerful in modeling network traffic and protocols, they do not have explicit support on hardware processing. Network emulators [13, 15, 19, 23] employ real hardware to represent network nodes. The modeling is accurate but lacking flexibility. We thus design a hardware aware network platform to provide highly flexible hardware modeling capability. As in Chapter 2, the platform adopts a network-hardware co-simulation approach in which the NS-2 network simulator supervises the network-wide traffic flow and the SystemC hardware simulator simulates the underlying hardware processing in network nodes. In addition, as a case study, we implemented wireless all-to-all broadcasting with network coding on the platform. We analyze the hardware processing behavior during the algorithm execution and evaluate the overall performance of the algorithm. Our experimental results demonstrate that hardware processing can have a significant impact on the algorithm performance. We expect that this hardware-aware platform will become a very useful tool for more accurate network simulations and more efficient design space exploration of processing-intensive applications.

1.2 Algorithm Design in Delay-Sensitive Wireless Sensor Networks

Recent years have witnessed the development and proliferation of wireless sensor networks (WSNs), attributed to the technological advances of Micro-Electro-Mechanical Systems (MEMS) and wireless communications. While early WSNs mainly focus on energy efficiency, delay sensitive WSNs have extra requirements on minimizing the communication delay during data delivery. Recent study in this area has mainly focused on the algorithm design of efficient routing strategies and data aggregation to reduce such delay and provide real-time delivery guarantees [29–32].

With the technology development, previous technologies whose original purposes are for energy efficiency are now adopted in delay-sensitive applications. The first technology we consider is compression. In WSNs, compression reduces the data amount by exploiting the redundancy resided in sensing data. Such redundancy comes from the strong spatial and temporal correlation existed in the sensor network [39]. Compress includes lossy compression and lossless compression. Lossy compression can achieve very high compression ratio by presenting only an approximate profile of the data [45–48]. A basic example of lossy compression is to provide only statistics (e.g. average) of readings from all sensors. Lossless compression, on the other hand, can reconstruct the original data from the compressed data. With the needs to obtain more accurate data of the physical world, lossless compression tends to become more important in the research world.

Besides energy saving, compression also contributes to delay reduction as it reduces the packet length and thus communication delay. However, the limited computing resources at sensor nodes make the processing time of compression a nontrivial factor in the total delay a packet experiences and must be carefully examined when adopting compression. In Chapter 3, we first study the effect of compression on data gathering in WSNs under a practical compression algorithm. We observe that compression does not always reduce packet delay in a WSN as commonly perceived, whereas its effect is jointly determined by the network configuration and hardware configuration. Based on this observation, we then design an adaptive algorithm to make on-line decisions such that compression is only performed when it can benefit the overall performance. We implement the algorithm in a completely distributed manner that utilizes only local information of individual sensor nodes. Our extensive experimental results show that the algorithm demonstrates good adaptiveness to network dynamics and maximizes compression benefit.

Clustering is another technique that originates from energy concerns. A typical cluster-based network is divided into two or more hierarchy. In the lower hierarchy, nodes are divided into clusters where a node acts as the cluster head to collect data from other cluster members. In the upper hierarchy, cluster heads communicate with each other and send collected packets to a data sink. While cluster heads take part in much more transmissions than cluster members and have faster energy consumption, they are either rotated within the network as in homogeneous clustering [67, 68] or assigned to devices with more power supply as in heterogeneous clustering [69, 85, 86].

Clustering is also known due to its good scalability as it simplifies the communication pattern. However, in such networks, frequent interactions between the intra-cluster communication and the inter-cluster communication are inevitable, which may severely downgrade the communication efficiency and hence the network performance if not handled properly. Proper synchronization among these two types of communications is required. In Chapter 4, we propose two approaches to schedule the communications in clustered wireless sensor networks aiming at delay-sensitive applications. In the first approach, an efficient cycle-based synchronous scheduling is proposed to achieve low average packet delay and high throughput by optimizing the cycle length and transmission order. In the second approach, a novel clustering structure is introduced to eliminate the necessity of communication synchronization so that packets are transmitted with no synchronization delay, yielding very low end-to-end packet delay. Our extensive experimental results demonstrate the superior performance of both approaches. These two approaches are then integrated as a hybrid scheme which allows smooth switching between them. The hybrid scheme takes advantage of both approaches and enables cluster-based sensor networks to serve as the fundamental network infrastructure for information collection.

Chapter 2

Construction of A Flexible Platform for Hardware-Aware Network Experiments

In this chapter, we present the design and implementation of a general, flexible hardware-aware network platform which takes hardware processing behavior into consideration to accurately evaluate network performance. We describe the synchronization and communication issue during the co-simulation between the network simulator NS-2 and the hardware simulator SystemC. To verify our motivation and illustrate the platform usage, we study all-to-all broadcasting based on network coding using our platform.

The chapter is organized as follows. Section 2.1 gives the background introduction. Section 2.2 introduces the related work in network simulation and emulation. Section 2.3 gives the detailed description of the platform architecture and the implementation. Section 2.4 describes a case study on the platform that evaluates the performance of wireless all-to-all broadcasting with network coding, and Section 2.5 presents the experimental results of the case study. Finally, Section 2.6 concludes the chapter.

2.1 Introduction

As network capability is improved by new technologies in terms of bandwidth, latency and services, many emerging applications, such as video-on-demand services and voice-over-IP, are now running on networks. Such applications put tremendous demand on network resources and also pose challenges on hardware to process a large volume of traffic at a high speed. In the meanwhile, more and more sophisticated network algorithms, such as header compression, packet aggregation and encryption, are introduced into network protocols to improve network performance, which greatly increases the complexity of hardware processing as well. Although complex hardware processing may be necessary to realize certain network functions, it has a significant impact on network latency, bandwidth and power consumption. This effect may vary among different hardware and network configurations. In general, computational resource limited network devices, including many mobile devices, are more prone to be affected by such complex processing. As a result, hardware-aware algorithm designs and designated hardware components may be required for such devices to optimize the overall performance of the network system. However, traditional network performance evaluation tools usually do not posses hardware-aware capability and it is difficult to use such tools to identify hardware performance bottleneck. Thus, a platform with hardware awareness is needed to study the effect of hardware processing in networks.

Currently, most network algorithms are developed, validated and evaluated by either simulation or emulation before their implementations are incorporated into practical products. Simulators, generally focusing on network traffic and packet transmissions, provide limited modeling capability for hardware processing. In particular, processing functions are implemented at software level and the hardware behavior is usually simplified as a fixed delay. Such modeling is apparently inadequate and inaccurate when the processing is complex and unpredictable, potentially leading to large disparity in network performance evaluation. Alternatively, emulators perform real-time simulations by performing the nodal processing and network transmissions with physical computers and network systems, thus they can obtain more accurate results on the behavior and performance of the simulated network. In traditional network emulators, the nodes in the simulated network are represented by actual hardware on a one-to-one or one-to-many mapping. In the case of one-to-one mapping, where each network node occupies one computer or network device, the processing is performed in hardware. Although this is the most realistic network environment, the network size in the emulation is strictly constrained by the number of hardware systems. In the case of one-to-many mapping, one or more network nodes are simulated as virtual nodes in one computer system, allowing certain scalability on the network size. However, since many virtual nodes share the same hardware utility, such as CPU or memory, hardware processing may not be realistically simulated. Besides, in both cases, the hardware systems used to represent network nodes have fixed hardware configuration and little hardware reconfigurable ability, which does not allow emulators to examine network performance under different hardware conditions. Moreover, some emulations may require certain expensive computational resources, which may not be always available or affordable.

From the above discussions, we can see that simulation provides inadequate supports for processing modeling, while emulation may be expensive and inflexible or not scalable. Thus, neither are suitable as a general platform to study the impact of hardware processing.

In this chapter, we present a general, flexible network platform for hardware-aware network simulations. The platform seamlessly integrates a network simulator to simulate the network environment and a hardware simulator to simulate the nodal hardware operations above the physical layer. Such integration of two simulators with different simulation levels is generally called *co-simulation*. In our case, the *network-hardware co-simulation* provides a complete experimental platform where hardware processing can be easily modeled and examined in a specific network scenario with sufficient flexibility of reconfiguring both the network infrastructure and the underlying hardware architecture. We also present a case study on wireless all-to-all broadcasting with network coding on this platform. In the case study, we describe the detailed procedure to perform the co-simulation. Our experimental results reveal that the processing time incurred by network coding has a significant impact on the overall performance, which increases the broadcasting time up to six times in the worst case. Due to its easy access to accurate, detailed information on hardware operations, which cannot be captured by the pure network simulation, and its flexibility of reconfiguring hardware components, which the network emulation lacks, we believe the platform will be a very useful tool for network algorithm/protocol designers and network related hardware designers.

2.2 Previous Related Work

As discussed earlier, two commonly used approaches to evaluating network performance are simulation and emulation. One representative of network simulators is NS-2 [2], which is widely used by the networking research community due to its good modeling capability for a variety of network scenarios and protocols. However, network simulators can provide only very limited modeling capability for hardware processing. NS-2 provides a primitive way to model hardware processing as delays by its timer mechanism. More sophisticated hardware modeling is out of consideration and difficult to implement in NS-2. Another powerful simulator OPNET [1] employs the process editor to model hardware processing using a state-transition diagram approach. However, it cannot model detailed hardware processing either.

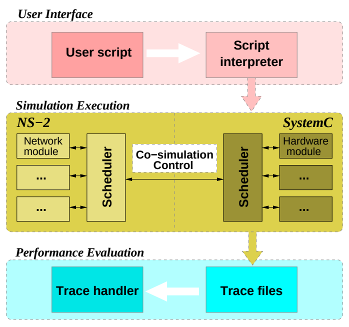
In addition, a simulator may be combined with some hardware facility to carry out network emulations for more realistic network modeling. Compared to pure network emulation, such combination aims to take advantage of resourceful network models in the simulator. For instance, NS-2 has been extended to have the emulation facility as described in [9]. The extension enables the simulator to interact with live network traffic, which may be generated by actual hardware. However, this approach requires the users to provide the desired real-time traffic or hardware, thus cannot serve as a general network emulator. In addition, the difficulty in synchronization with real-time traffic prohibits the entire emulation from being performed at high speed.

On the other hand, network emulators employ actual computer systems and networks to provide a more realistic network environment. In the emulation, physical nodes represent the network nodes and a virtual network is created to regulate the network topology and protocols among all the nodes. Emulators for general networks, see, for example, [5, 6, 14, 21, 22], usually adopt real PCs as the physical nodes. Then hardware processing is actually executed rather than simulated, yielding more accurate results. However, the real execution implies that the processing is tightly associated with the hardware configuration of the physical nodes and may become inaccurate if the hardware configuration changes. There are also network emulators for a specific type of networks, such as Emulab [15], Motelab [19], ORBIT [23] and Emstar [13]. For example, the 802.11a/b/g testbed in Emulab scatters two different types of nodes with real wireless interfaces in an office building, allowing a number of configuration parameters. Compared to Emulab and other similar testbeds, which aim at constructing a specific real network environment, our proposed platform emphasizes on supporting general, flexible hardware modeling, which is usually neglected in other simulators and emulators.

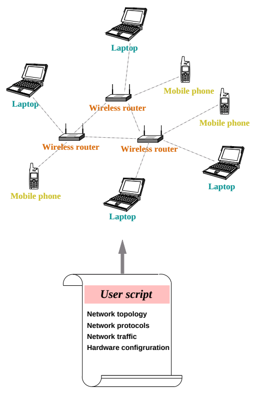
Finally, a simple network-hardware co-simulation method was adopted in [12] for a specific simulation environment with networked embedded systems. Its main focus is to study the behavior of embedded systems when they are connected to a practical network environment. In such a system, there is a natural partition between the network modeling and the hardware modeling, which makes co-simulation much easier. However, it is difficult to use such an approach for general networks as there may not exist such a clear interface between the network and the hardware. In this chapter, we will focus on general network-hardware co-simulation and treat the network traffic and the hardware nodes as an integrated system. In such a system, any network protocols and algorithms can be modeled in hardware so that network performance can be studied in a more realistic environment.

2.3 Platform Implementation

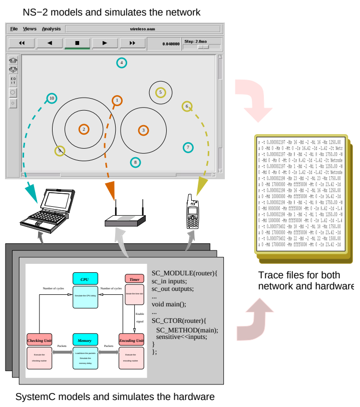
In this section, we describe the implementation of our network platform, which is based on network-hardware co-simulation. In the co-simulation, we adopt NS-2 simulator as the network simulator and SystemC hardware simulator as the hardware simulator. NS-2 is a popular software-level network simulator which is capable of creating various networks with flexible configurations; SystemC is a system-level hardware description language which can describe the specific hardware without going into the details of the underlying hardware implementation. At the network level, NS-2 manages the global network scenario and sends the packets to SystemC for hardware simulation when necessary. The interface to SystemC is similar to the interface of an ordinary network module so that NS-2 can acquire the hardware processing ability without significant changes of its own framework. At the hardware level, SystemC models and simulates the specific hardware processing taking the network traffic as its input and returns the processed packets to NS-2. With co-simulation, two simulators cooperate and work as a single simulator performing both network and hardware operations.



(a) Platform overview.



(b) User scripts define the desired network environment.



```
SC_MODULE(router){
sc_in inputs;
sc_out outputs;
...
void main();
...
SC_CTOR(router){
  SC_METHOD(main);
  sensitive<<inputs;
}
};
```

(c) NS-2 and SystemC simulate the network and hardware respectively.

Figure 2.1: The overview of the platform and an example showing its usage.

Fig. 2.1(a) illustrates the overview of our platform. Similar to most network simulators, the platform requires a script provided by the user to specify the network environment. Such specification includes network topology, network traffic, protocols used in the network and interfaces to hardware models. The user script is written in OTcl, the language used in NS-2, to provide a convenient user interface considering NS-2's current popularity. Fig. 2.1(b) shows an example where the user script specifies a wireless network consisting of wireless routers, laptops and mobile phones. The script is interpreted directly by NS-2 and accordingly, NS-2 and SystemC will perform the network and hardware simulation cooperatively. As shown in Fig. 2.1(c), the devices in the wireless network are reduced to uniform network nodes which perform packet transmissions in NS-2 while the hardware processing in the network devices is concurrently executed in SystemC. At the end of the simulation, both simulators generate simulation trace files, which are returned to the user for performance evaluation and potential optimization on the hardware design.

Before we go into the details, we first briefly discuss why it is necessary to use co-simulation to incorporate hardware modeling capability in the network simulation. A possible alternative is to implement hardware modeling as a function call when it is necessary to simulate hardware processing. The processing results and delay can then be calculated and returned to NS-2. Unfortunately, such an approach is only applicable when hardware processing is rather simple and independent of network traffic. The traffic can affect both processing results and delay. For example, in packet compression, when a new packet is generated, the ongoing compression process may be prolonged due to the addition of this new packet, changing both compression results and delay. As will be seen in our case study later, multiple processes executed in a single processor can affect the completion time of each process. Therefore, it is necessary to perform co-simulation in order to obtain a complete hardware modeling capability.

In the co-simulation, NS-2 and SystemC are executed separately in two processes due to their different simulation behaviors. As shown in Fig. 2.2, three major issues need to be addressed in the co-simulation implementation. First, as each simulator has its own simulation timer, time synchronization of the two simulators must be maintained when packet exchanges between two simulators occur. Second, inter-process communication must be handled efficiently so that the communication overhead would not affect the simulation speed. Third, to practically perform the co-simulation, the interfaces between user modules and the schedulers in both simulators must be extended to provide special functions for co-simulation. Next we discuss these three issues in detail.

2.3.1 Time Synchronization

Time synchronization is a critical issue in the co-simulation. As both NS-2 and SystemC are event-driven simulators, each simulation is managed by the scheduler, which schedules and executes all the events in the order of time.

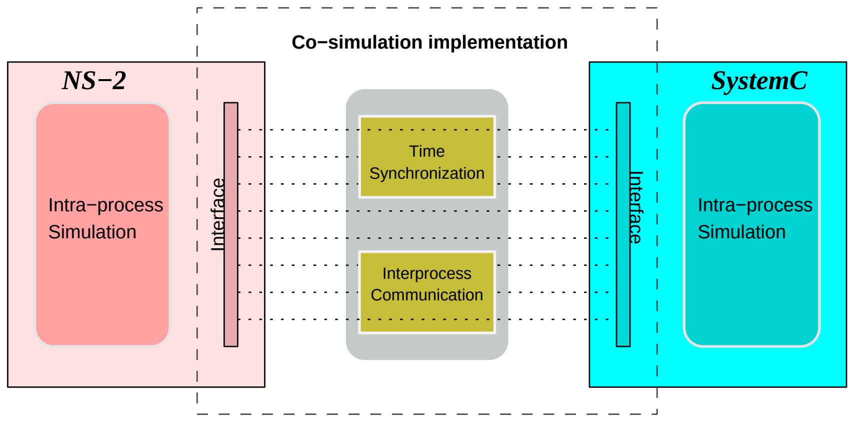


Figure 2.2: Three major issues in the co-simulation implementation: time synchronization, interprocess communication and module-scheduler interface design.

In the co-simulation, the events must be scheduled and executed by a global scheduler with a global timer. However, it is difficult to implement a global scheduler because the schedulers of the two simulators have different specifications on the events. Our solution is to insert a synchronization mechanism in both schedulers so that they can schedule their own events and in the meanwhile maintain the time synchronization between them.

Besides the events originally generated in the two simulators, the co-simulation incurs another type of new events, that is, the events of sending messages between the two simulators. For presentational convenience, we classify the events in the co-simulation into two categories: *Type I* events, the original events in the two simulators and *Type II* events, the events of sending messages between the two simulators. Each event is associated with a time stamp indicating the exact simulated time this event is executed, called the *time of the event*. The event with the earliest time in the co-simulation is called the *executable event* and there could be multiple executable events. Only executable events can be executed and the simulation time advances to the time of the executable event when executed. In each simulator, the scheduler maintains the list of events locally generated, among which the earliest Type I event is called the *schedule event* and its time is called the *schedule time* of the simulator. By definition, a Type II event is generated when a simulator decides to send a message to the other simulator, during the execution of a Type I event. Therefore, the time of the generated Type II event is the same as the Type I event currently being executed. Moreover, since this Type I event must be an executable event, all Type II events generated during its execution become executable events instantly.

At the beginning of the co-simulation, both simulators have only Type I events. After exchanging the schedule time between the two simulators, the simulator with earlier schedule time starts the simulation while the other simulator pauses and waits for the messages. Executable Type I events are then executed according to the local scheduling strategy until there are no more such events, which means that the schedule time of this simulator surpasses either the schedule time of the other simulator or the time of Type II events. Then all the Type II events (if any) are executed and the messages are sent to the other simulator. Notice that in this case Type I events are executed prior to Type II events with the same time stamp. In fact, neither NS-2 nor SystemC defines the execution order of events with the same simulation time, so such scheduling does not violate the principles of original schedulers. On the other hand, since each execution of Type II events leads to a message transmission and a control switching between the two simulators, putting all the Type II events together can minimize the inevitable time cost for communication setups. After executing Type II events, the current simulator completes its execution cycle and pauses while the other simulator begins its execution cycle by receiving the messages. The execution cycles will be performed in turn among two simulators until no more events are left in either simulator. Then the co-simulation terminates. Table 2.1 gives the pseudo code of the event scheduling algorithm.

As shown in Table 2.1, two processes corresponding to the simulations of NS-2 and SystemC are executed concurrently. In NS-2 process, one while loop (line 5) does the scheduling and another while loop (line 7) checks if there are any executable events. The generation of Type II events is implicitly implemented in the event execution (line 12) and the implementation detail will be described in the next section. If there exist executable Type I events, the scheduler will pick up an event to execute (lines 12 through 13); otherwise, the communication procedure starts by calling functions *send_to*/*wait_from* (lines 8 through 10). Function *send_to* executes all Type II events and sends corresponding messages to the other simulator. After that, it sends the local schedule time for synchronization purpose. Function *wait_from* will force NS-2 to wait until receiving the message from SystemC. Then it calls the corresponding functions to process the messages. Message processing is similar to the execution of an event, thus local simulation time t_1 could be correctly updated after the message processing. Since we always execute the executable events which have the earliest executing time, the synchronization is strictly guaranteed. The scheduling in SystemC process is similar. The main difference is that SystemC process will yield to NS-2 process when both schedulers

Table 2.1: Scheduling Algorithm in NS-2 and SystemC

```
1.   $t_1$ = schedule_time(NS-2);
2.   $t_2$ = schedule_time(SystemC);

3.   **NS-2:**
4.   wait_from_SystemC($t_2$);
5.   while(exist(event)){
6.       update($t_1$);
7.       while($t_1 > t_2 || t_1 >$ time(Type II events)){
8.           send_to_SystemC(Type II events);
9.           send_to_SystemC($t_1$);
10.          wait_from_SystemC($t_2$);
         }
11.      get executable event e;
12.      execute(e);
     }

13.  **SystemC:**
14.  send_to_NS-2($t_2$);
15.  wait_from_NS-2($t_1$);
16.  while(exist(event)){
17.      update($t_2$);
18.      while($t_2 \geq t_1 || t_1 >$ time(Type II events)){
19.          send_to_NS2(Type II events);
20.          send_to_NS-2($t_2$);
21.          wait_from_NS-2($t_1$);
         }
22.      get executable event e;
23.      execute(e);
     }
```

have the same schedule time to avoid the potential deadlock in this situation.

2.3.2 Interprocess Communication

As mentioned earlier, Type II events send messages between the two simulators. How to efficiently communicate between the two simulators executed in two processes is another critical issue in the co-simulation.

Since NS-2 and SystemC are executed in two processes, their communication is accomplished through an interprocess queue. Fig. 2.3 illustrates the communication model between NS-2 and SystemC, where the entire procedure can be divided into three phases: message buffering, message transmission and message handling.

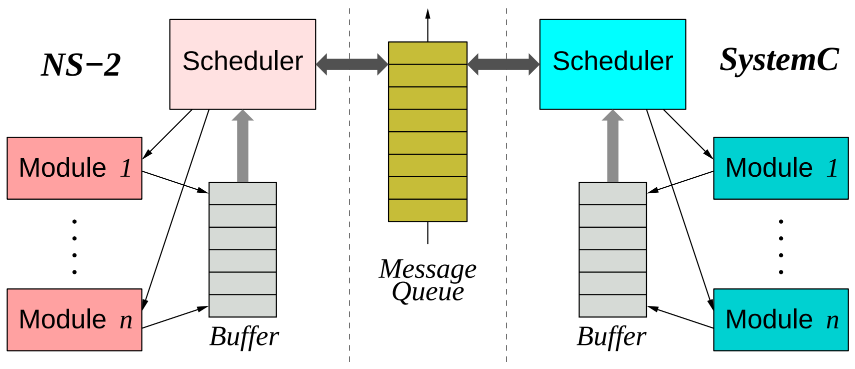


Figure 2.3: Communication model in the co-simulation, where a module corresponds to a functional unit in a network node. The communication is divided into three phases: message buffering, message transmission and message handling.

As mentioned in the last subsection, Type II events are generated during the execution of Type I events. The creation of Type II events implies a message transmission between the two simulators. As shown in Table 2.1, instead of starting the message transmission immediately, the scheduler will wait until all the executable Type I events are executed. Therefore, a message will be temporarily stored in the buffer whenever a Type II event is generated during the execution.

Message transmission occurs when there are no more executable Type I events in the scheduler. Since the two simulators reside in different processes, messages cannot be sent directly. We use the message queue as the interprocess communication mechanism so that messages can be conveniently and efficiently sent and managed. The message queue is responsible for receiving messages from one simulator and sending messages to the other simulator.

To complete the message transmission, some additional information should be attached to the actual message. The message format used in the co-simulation includes four fields: *mid, length, time* and *data*. *mid* specifies the destination module this message should be delivered to, where a module corresponds to a functional unit in the network node. To correctly receive the messages, every module in SystemC must have a corresponding module in NS-2 and the two modules are assigned the same *mid* before the simulation. *length* indicates the length of field *data*, facilitating the receiving of messages. *time* records the time when the message is created. When the simulator on the receiving side receives this message, the simulation time will be advanced according to *time*. *data* contains the actual message stored as a string. The receiver is responsible for converting the string to the proper data format.

The receiving procedure does not require any buffer. The messages received will be immediately delivered to the corresponding module indicated in *mid*. Since both simulators are event-driven, message delivery is considered as the execution of an event.

2.3.3 Module-Scheduler Interface Design

In both NS-2 and SystemC, the scheduler schedules the events, while other functional modules execute the events. The synchronization and communication are mainly accomplished by the scheduler, and message buffering and message handling require the cooperation between the scheduler and functional modules. The interface between the scheduler and functional modules must be extended to provide extra functions for the co-simulation. In particular, the scheduler should provide the store function for functional modules to temporarily store the messages in the local buffer while functional modules are required to provide corresponding handlers to handle the receiving of messages. Due to the differences in the frameworks of two simulators, we describe the interfaces in NS-2 and SystemC separately next.

NS-2 Interface

In NS-2, the communication among different modules is realized by calling functions *send/recv* which send/receive data packets between the modules. Two similar functions *sendmsg/recvmsg* are provided to process the co-simulation messages. Function *sendmsg* is provided and implemented by the scheduler. During the execution of a type I event, once a Type II event is generated, a formatted message is created and passed as an argument for function *sendmsg* to run. Function *recvmsg* is provided by functional modules while its interface is added as a virtual function in the base class, from which all the functional modules in NS-2 inherit, so that the scheduler can call this function in different modules. The implementation of *recvmsg* is module dependent, thus, each co-simulation module expecting the reception of message needs to define its own implementation of *recvmsg*.

SystemC Interface

In SystemC, a special module ns_module is created to communicate between the scheduler and other modules. In the simulation, ns_module can be considered as a regular module which runs NS-2 simulator, thus the communication between the two simulators becomes the communication between ns_module and other modules. In SystemC, channels are the communication media and modules use ports to access the channels. In the implementation, two special ports *send/recv* are created as the interface between ns_module and other modules.

ns_module communicates with NS-2 through functions *sendmsg/recvmsg*. Different from NS-2, SystemC does not allow the immediate processing of the received message. Therefore, when function *recvmsg* is called by the scheduler, ns_module will store the message in its buffer and send a processing request to the scheduler. When the request is granted, ns_module will send the message to other modules through port *recv*. When other modules send a message to ns_module through port *send*, ns_module will call function *sendmsg* to send the message to the buffer.

2.3.4 Platform Usage and Summary

The proposed platform can greatly benefit the design space exploration and early stage optimization. As shown in Fig. 2.4, in a conventional design without such a platform, a network protocol or algorithm is initially designed and evaluated by network simulation to obtain network level performance. After the evaluation, the algorithm will then be implemented in a system testbed to validate the practical performance including both network performance and hardware performance. With only evaluations at the network level, the design may not correctly consider the impact of hardware, which leads to performance disparity. As a result, modifications are needed to accommodate the system requirements and such modifications may have to be performed on the testbed due to the inaccuracy of the simulation. However, hardware level modification and further validation on the testbed are generally time consuming and difficult. In the worst case, the entire process may be degenerated to a testbed design/implementation process only, invalidating the original design obtained by network simulation.

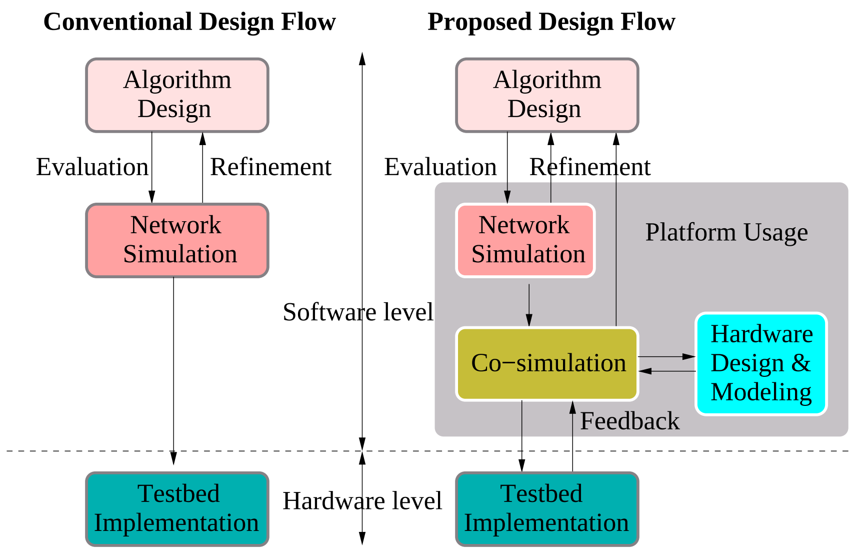


Figure 2.4: The conventional design flow vs the proposed design flow with the platform.

We can see the gap between the network simulation and the testbed implementation prevents the efficient interaction between network and system level designs. Our platform can perfectly fill this gap. The algorithm design can be directly evaluated on the platform to perform either pure network simulation (by setting the processing delay to 0 at SystemC) or network-hardware co-simulation. With the hardware correctly modeled, the algorithm can be designed from a system perspective at the very early stage, reducing the potential modifications in testbed implementation. On the other hand, modifications can also be validated with co-simulation based on the adequate feedback from the testbed. This way, the whole design process becomes much more flexible and efficient with the support of the platform.

To utilize the platform, network protocols and algorithms should be converted into SystemC modules to simulate hardware processing. In practice, algorithms can be executed either in hardware or software. If an algorithm is executed in a specific hardware module, for example, an FPGA, hardware processing can be easily modeled given the strong hardware modeling ability of SystemC. When it is run in software on a processor, an accurate modeling of the execution becomes more complex and requires strong hardware modeling ability, causing a potential burden for network designers. Fortunately, techniques that automatically convert software into SystemC modules have been extensively studied [24–27]. Users thus can utilize such tools to generate the desired SystemC modules for the co-simulation. We will show an example of such modeling in the following case study.

With the SystemC modules constructed, we can perform network-hardware co-simulation to evaluate the system performance from both network perspective and hardware perspective. The evaluation results can then be analyzed to assist the optimization and refinement on the network protocol, underlying hardware system or a combination of them.

Finally, we summarize the primary features of the platform as follows.

1. The platform provides powerful modeling capability for both network and hardware through the co-simulation of NS-2 and SystemC.

2. The platform adopts NS-2 user interface, facilitating the easy use by a large number of users familiar with NS-2.

3. The co-simulation is mainly accomplished by the schedulers of two simulators. Since NS-3 has similar scheduling implementation as NS-2, the network simulator in our platform can be easily upgraded to NS-3 which is currently released but still under development.

4. The platform provides great flexibility in hardware modeling such that the users can flexibly partition simulated functions between hardware and software. Besides, a set of automation tools provided by the SystemC community facilitate the construction of hardware models for network protocols/algorithms on any system architectures.

5. The platform is executed purely in software, without any requirement on specific hardware support.

2.4 Case Study of Wireless All-to-All Broadcasting Using Network Coding

The co-simulation of NS-2 simulator and SystemC simulator provides a configurable environment for studying network behaviors with hardware awareness. In this section, we present a case study on the platform that goes through the entire procedure of the co-simulation to demonstrate its usage and capability.

We consider all-to-all broadcasting in a wireless network that uses network coding in packet transmissions. All-to-all broadcasting is the most bandwidth consuming communication operation that can reveal network performance in the most stressed scenario. Network coding is a promising generalization of routing that allows a network node to generate output packets by encoding its received packets to reduce bandwidth consumption in the network and improve network throughput. While network coding leads to improved throughput, due to the extra work in coding/decoding packets, it inevitably increases the processing complexity of network nodes as well. However, this type of processing overhead is difficult to accurately simulate in conventional network simulators, often leading to inaccurate performance results.

As the case study, we examine a network algorithm that implements wireless all-to-all broadcasting with network coding on our platform by taking hardware processing into consideration. The purpose of this case study is to 1) reveal the necessary details on conducting a network-hardware co-simulation, and 2) examine the impact of hardware processing on the performance of this network coding algorithm.

Next we first introduce the all-to-all broadcasting algorithm based on network coding and analyze its hardware processing by decomposing it into different functional units. Then we present the hardware modeling of intra-node processing on the platform. We will give the experimental results in Section 2.5.

2.4.1 Wireless All-to-All Broadcasting with Network Coding

The concept of network coding was first introduced by Alswede, et al. in [4] to increase multicast capacity in wired multicast networks. The basic idea is that instead of using the traditional way to forward information flows at intermediate nodes, network coding combines independent information flows to better utilize network bandwidth and achieve higher throughput. Later, linear coding was introduced and proved to be sufficient to achieve the theoretical optimum of the information rate [7]. While much work has been done on finding optimal coding schemes, a random network coding scheme was also proposed in [8] to provide near-optimal performance and make network coding more practical.

Although earlier work on network coding was for wired networks, recently, this promising technique has been extended to wireless networks. By utilizing the characteristics of the broadcast channel, information in a wireless network is mixed and forwarded at intermediate nodes to improve network performance in terms of throughput, energy efficiency, etc. In [10] the benefit of network coding in wireless networks was first studied. Since then, many algorithms have been proposed to utilize network coding in both wireless unicast and multicast.

More recent work has been focused on the algorithms that can implement wireless network coding in practical networks. For example, random coding has been adopted in unicast and multicast to achieve energy efficiency and increase throughput [11, 18], while algorithms using opportunistic coding for wireless unicast and multicast were proposed in [16, 17].

Since this case study attempts to examine the processing impact on the network algorithm rather than to design a new network coding algorithm, we adopt the algorithm introduced in [11], where random network coding is used to achieve high efficiency on all-to-all broadcasting. This algorithm is the first distributed algorithm in its category and can be easily applied to a wireless network environment.

Consider a wireless ad hoc network with n nodes, each sending a packet to all other nodes. For simplicity, we assume the packets are of the same length. Let x_i denote the packet that node i needs to send. With network coding, the packet sent or received by a node is a linear combination y of the packets such that $y = \Sigma_{i=1}^{n} g_i x_i$, where g_i is the coding coefficient on x_i and vector $g = \{g_1, g_2, \ldots, g_n\}$ is the coding vector of packet y. In each transmission, a new packet and its coding vector are generated by randomly and linearly combining the existing packets and their coding vectors. The coding vector is sent along with the packet. If the coding vector associated with the incoming packet is linearly independent of all the coding vectors received earlier, the packet is called *innovative*. When a node receives n innovative packets, which means that the n corresponding coding vectors form a full rank $n \times n$ matrix, the original packets can be obtained by solving the following equation

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} g_{11} & \cdots & g_{1n} \\ \vdots & \ddots & \vdots \\ g_{n1} & \cdots & g_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}.$$

When there are multiple packets to send from each node, we group the packets into generations such that a generation contains a single packet from each node. The all-to-all broadcasting starts from the first generation and proceeds to the next generation when the previous one is successfully decoded.

The algorithm in [11] is mainly concerned with the energy consumption of all-to-all broadcasting by counting the number of transmissions performed to achieve broadcasting. Fewer transmissions usually cost shorter time and thus yield better throughput. However, to actually evaluate the algorithm in terms of throughput, several practical issues that were ignored in the original algorithm should be reconsidered.

The first issue is transmission scheduling. The algorithm requires that every node keeps broadcasting its coded packets to neighbor nodes until all packets are decoded. While the original algorithm adopts a TDMA-based approach, which may not be applicable in large networks, we adopt the commonly used MAC 802.11 in our experiment. At the network layer, we allow each node to generate a coded packet for transmission in a fixed time interval, which is used to reduce the impact of potential transmission collisions. In MAC 802.11, broadcasting packets are transmitted without acknowledgments, making transmissions unreliable. Although carrier-sensing can prevent neighbor nodes from concurrent transmissions, it cannot prevent non-adjacent nodes from concurrent transmissions due to the hidden-terminal problem. A short time interval may lead to severe transmission collisions and hence downgrade the performance. On the other hand, a long time interval would reduce channel utilization and hence the throughput. Therefore, we set the time interval to $(N_{nb}+1)(T_{tran}+D_{max})$, where N_{nb} is the number of neighbors, T_{tran} is the transmission time of a packet and D_{max} is the longest backoff delay before transmission. This is equivalent to the time interval when an ideal scheduling strategy fairly schedules all the transmissions that saturate the channel without collisions. Notice that such setting may not be the optimal strategy but serves as a necessary parameter for the evaluation, which focus more on the effect of processing than on the broadcast performance itself.

The second issue is to decide when each node should proceed to the next generation. A natural approach is that each node notifies the neighbors when the packets are decoded, then when a node has received the notifications from all its neighbor nodes, it can proceed to the next generation. However, since broadcasting in MAC 802.11 is unreliable transmission, there is no way to confirm if the notification is correctly received by neighbors. Our solution is to impose a limit on the maximum number of transmissions m_t and the minimum number of notifications m_n. Once a node decodes all the packets, it sends the coded packets along with the notification at least m_n times. After that, it continues to transmit until the node receives the notifications from all its neighbors or the number of transmissions reaches m_t. Thus m_t allows the node to proceed eventually while m_n allows neighbors to receive the notification with a high probability. The pseudo code of the enhanced algorithm is shown in Table 2.2.

2.4.2 Function Decomposition of Intra-node Processing

After describing the implementation issues of the algorithm, we now consider modeling the intra-node hardware processing in order to perform network-hardware co-simulation. Next, we decompose the intra-node processing into separate functions and describe the main operations of each function.

Table 2.2: Intra-node Procedure in All-to-all Broadcasting

```
Transmitting procedure:
while TRUE
    if the packets are decoded
        proceed to the next generation when permitted
    end if
    Generate a coding vector randomly
    Encode the packet
    Send the packet
    Wait for next time interval
end while

Receiving procedure:
while receive a packet
    if it is from another generation
        Discard the packet and wait for the next packet
    end if
    Store the notification of sender if any
    if the packet is innovative
        Store the packet in the buffer
    end if
    if the buffer has $n$ packets
        Decode the packets
        Enable notification and start the transmission counter
    end if
end while
```

In the all-to-all broadcasting algorithm, the intra-node processing can be divided into the following five routines based on their functions:

1. Receiving routine. The incoming packets are stored in a temporary buffer.

2. Checking routine. This routine checks whether the packets in the buffer are innovative. An innovative packet has a coding vector independent of the coding vectors already received. The dependency check is accomplished by checking whether the rank of the matrix composed of the coding vectors increases when the incoming coding vector is added to the matrix. If a packet is innovative, it will be stored in the receiving buffer; otherwise, the packet will be disposed immediately.

3. Encoding routine. When a node is scheduled to transmit, a new encoded packet will be created by performing a random linear combination among all the packets in the receiving buffer of the node.

4. Transmitting routine. The prepared packet will be broadcast by the node.

5. Decoding routine. This routine starts when n innovative packets are received, where n is the generation size. The decoding routine is similar to the checking routine except that Gaussian Elimination is performed on both coding vectors and the packet data.

We assume that the receiving and transmitting routines are performed by specific transceiver, and their processing times are mainly determined by the network transmission and are unnecessary to be considered in the hardware modeling. Besides, the execution of the decoding routine occurs only when all packets in a generation can be decoded and thus is ignored due to its relatively small impact on the broadcasting time. In the following analysis, we consider the checking and encoding routines.

The checking routine computes the rank of the code matrix by Gaussian Elimination. Given generation size n, the matrix size is at most $n \times n$. A primitive implementation of checking routine will take $O(n^3)$ time, while the encoding routine encodes a packet in $O(n(n+P))$ time, where P is the packet length. To speedup the process, we adopt the optimization technique proposed in [18]. For the checking routine, the code matrix is stored in row echelon form so that the time complexity is reduced to $O(n^2)$. A new encoded packet is always pre-computed when the MAC layer is transmitting the last packet to minimize the processing delay. When an innovative packet arrives, the pre-computed packet should be updated by another encoding routine, which takes $O(n+P)$ time.

2.4.3 Hardware Modeling for Intra-node Processing

Now we describe the hardware modeling for intra-node processing. We construct a SystemC module representing the hardware system of the node as shown in Fig. 2.5. Such a module will be created for each node in the simulation and consists of four units. The ns_module as already mentioned in Section 2.3, takes charge of the packet exchange with NS-2. The checking and encoding units simulate the hardware execution of two routines, whose behavior is determined by the particular system architecture of the network node. The os_module manages the scheduling of multiple tasks in the system. As can be seen, the modeling focuses on the behavior of checking and encoding routines as they have a major impact on the algorithm performance. To realize co-simulation, we also need to construct a connecting point in NS-2 to incorporate the hardware module in the network simulation. Since the two routines are at network layer, we construct a connecting network layer agent in each node. The agent inherits the same interface used in other agents and its main function is to exchange the packets between other modules in NS-2 and the SystemC module. From the network perspective, the agent can be considered as an ordinary NS-2 agent that implements the network coding algorithm with accurate hardware timing.

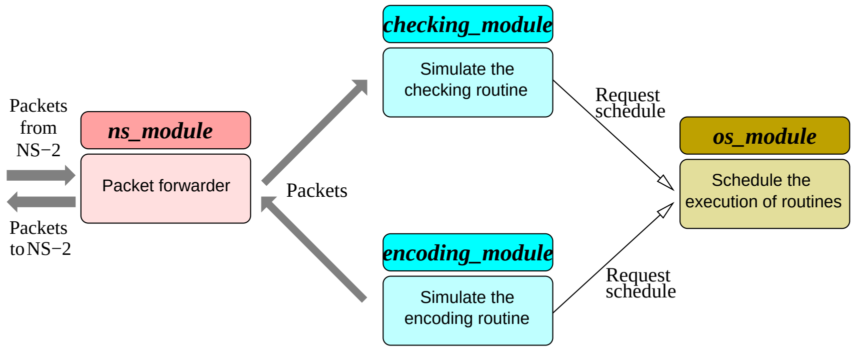


Figure 2.5: Structure of SystemC module simulating intra-node hardware processing.

To achieve accuracy in hardware simulation, the modeling of the two functioning routines is very important. As a hardware description language, SystemC can easily model the processing if routines are implemented in some designated hardware. Here we consider that routines are executed at software level on a MIPS processor, which is widely used in commercial wireless routers such as Cisco Linksys series.

The SystemC community has provided a variety of modeling techniques to model the processing. The most accurate approach is the utilization of Instruction Set Simulator (ISS) to simulate the hardware operation of the MIPS processor. The routines are executed in the ISS the same way as in the real processor. While this approach can be quite time consuming, a much faster alternative adopted in our work is to use SystemC's transaction level modeling (TLM) to estimate the execution time of each routine during the software level execution. This estimation is accomplished by annotating the source code of the routine with associated timing delta for each source line. The reported estimation accuracy is generally above 90% [24–26].

The approach we adopt is similar to the one described in [25], thus we give only a brief introduction here. To obtain the timing delta of the source code, we first cross-compile the source code into the binary code for MIPS with GNU Compiler Collection (gcc). For each source line, we find the corresponding binary code and estimate the execution cycles based on the pipeline structure, instruction issue mechanism, cache behavior, branch prediction, etc. As the binary code can be directly executed on the target processor, such estimation provides very high accuracy with a correct description of the architecture. With the estimation, instrumentation code is then inserted in the source code to simulate the hardware delay. For the efficiency consideration, the insertion is performed once for each basic block, in which statements are sequentially executed without conditional branches. The instrumented source code is then wrapped in a SystemC module, which represents the hardware module for the corresponding routine.

In our example, the target processor is a MIPS32 M14Kc core with a 5-stage pipeline. It is a scalar processor with a fixed 1-cycle branch penalty. For simplicity, we omit the cache behavior simulation in the cycle estimation. Nevertheless, such estimation provides an accuracy of above 93% as reported in [25].

In the network algorithm, the execution of checking and encoding routines could overlap, which requires the modeling of multiple task scheduling. We treat the routines of checking or encoding a packet as an individual task and construct os_module to schedule the task executions. Specifically, in the SystemC modules that simulate the routines, the original delay simulation statements are replaced by a function call to os_module with the requested delay passed as a parameter. os_module that maintains the status of all running modules will then decide when to process the requests based on the scheduling policy.

2.5 Experimental Evaluation

In this section, we present the experimental results for the wireless all-to-all broadcasting algorithm with network coding and analyze the impact of hardware processing on its overall performance. In addition, we examine the execution time of the co-simulation running this algorithm to better understand the runtime performance of the platform.

2.5.1 Experiment Setup

We evaluate the broadcasting algorithm in two networks: a 5×5 grid with 25 nodes and a 6×6 grid with 36 nodes. The distance between neighboring nodes is $10m$ and each node can broadcast only to its neighbors in the grid. The channel bandwidth B is set to be either $54Mb/s$, $11Mb/s$ or $2Mb/s$ to represent different network conditions. The MAC layer adopts simplified MAC 802.11 where a random delay is inserted before the transmission when the channel is sensed idle. Packets are of a fixed length, which is set to be $100B$ or $1000B$ to reveal the impact of the packet length.

For hardware modeling, we assume that each node has a MIPS32 processor as we previously modeled with CPU frequency at $240MHz$, a typical frequency used in System-on-Chip (SoC) for routers. The task scheduling policy is non-preemptive if not specified otherwise.

In the algorithm, m_t is set equal to number of nodes and $m_n = 4$. The performance metric is the time when ten generations of packets are successfully received by all nodes in the network. Each experiment is repeated 10 times and the average is used for evaluation.

2.5.2 Verification of Synchronization Mechanism

We verify the synchronization mechanism in co-simulation by comparing the simulation results of the broadcasting algorithm in only NS-2 simulation with those in co-simulation on the proposed platform with the processing delay set to 0. For fairness consideration, we use the same seed for random number generation. The results show that both approaches complete the broadcasting with the same finish time, showing that co-simulation will not disturb the network behavior with the absence of processing delay.

2.5.3 Impact of Hardware Processing

To illustrate the impact of hardware processing on the overall performance, we compare the simulation results when processing delay is not considered and when the hardware processing is simulated in the co-simulation. Fig. 2.6 depicts the performance comparison under different network configurations.

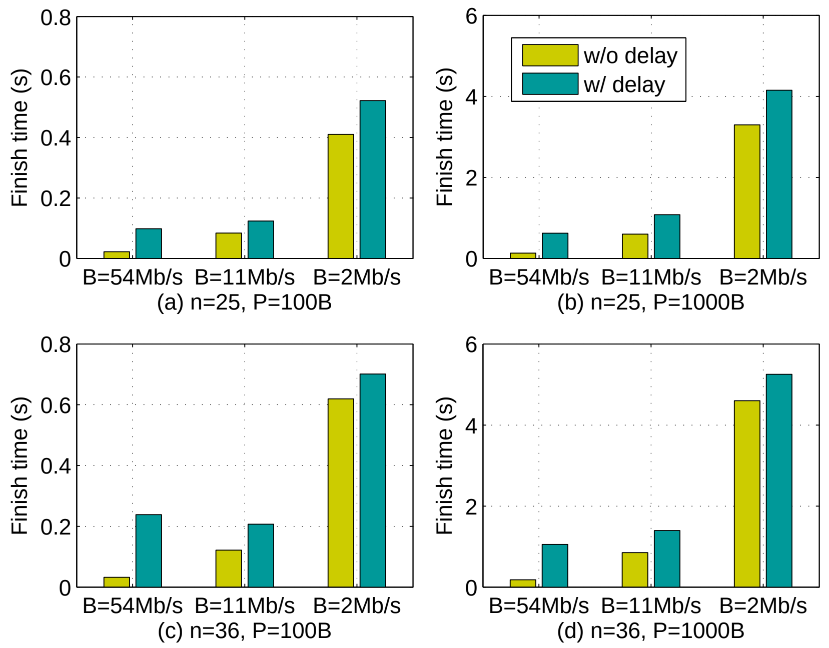


Figure 2.6: Comparison of broadcasting finish time with different network configurations, where n represents the number of nodes or the generation size, P is the packet length and B is the channel bandwidth.

In Fig. 2.6, the finish time without processing delay is approximately in reverse proportion to the bandwidth, which in turn is reversely proportional to the packet transmission time given the fixed packet length. Therefore, we can infer from such correlation that the number of transmissions in the broadcasting does not vary a lot with different transmission speeds so that the finish time is mainly determined by the packet transmission time or the bandwidth.

On the contrast, when the processing delay is simulated, although the finish time decreases with the increase of the bandwidth, the reverse proportion does not hold. For example, in Fig. 2.6(a), the finish time drops less than 20% when the bandwidth increases from $11Mb/s$ to $54Mb/s$, which indicates a varying impact of processing delay on the overall performance with different bandwidths. In fact, the processing delay causes an increase of 27%, 47% and 345% on the finish time when the bandwidth is $2Mb/s$, $11Mb/s$ and $54Mb/s$, respectively. Thus, the processing delay can severely degrade the broadcasting performance.

Fig. 2.6 also shows the performance comparison under different generation sizes and packet lengths. We see that the processing impact is relatively steady under different configurations except for the case when $B = 54Mb/s$. Higher impact can be observed with a shorter packet length and a larger generation size, both contributing to relatively higher overhead of the coding vector and longer execution time of the checking routine. The highest impact can be found at the configuration with $P = 100B, n = 36$: the finish time even exceeds the value with a lower bandwidth $11Mb/s$. This corresponds to the fact that such configuration has the longest execution time of both checking and encoding routines, which is even longer than the time interval. Thus, the broadcast finish time is mainly determined by the processing instead of the time interval. In this case, a smaller time interval may generate packets before the received packets are examined, reducing the percentage of innovative packets and hence the broadcast performance.

2.5.4 Processing Delay

To better understand the effect of processing time, we examine the processing time of both encoding and checking routines in the whole simulation for a node in the center of the grid, as plotted in Fig. 2.7. The two cases we examine are $B = 2Mb/s, n = 25, P = 100B$ and $B = 54Mb/s, n = 36, P = 1000B$. The corresponding time intervals are $2550\mu s$ and $850\mu s$ respectively as the node has 4 neighbors. As can be seen from the figure, the execution time increases when the number of innovative packets in the nodes increases and such pattern is repeated 10 times, corresponding to the broadcasting of 10 generations.

Specifically, the execution time of the encoding routine is much longer than that of the checking routine since the encoding routine depends on both packet length and generation size while the checking routine depends only on the latter. In addition, there are also some "abnormal" points in Fig. 2.7 that do not match the complexity analysis, which is due to the contention in the use of CPU. With a single CPU adopted in the node, routines have to wait for the completion of the ongoing routine in the CPU, prolonging the entire execution time. Typically, we see there are more obviously abnormal points for the checking routine in Fig. 2.7(b), as in this case the encoding routine has a relatively longer execution time, increasing the possibility of contention for the checking routine.

In Fig. 2.7(a), we see the maximum execution time is around $600\mu s$, shorter than the time interval $2550\mu s$. However, we still see a 20% performance degradation. This indicates that the execution of both routines always incur certain

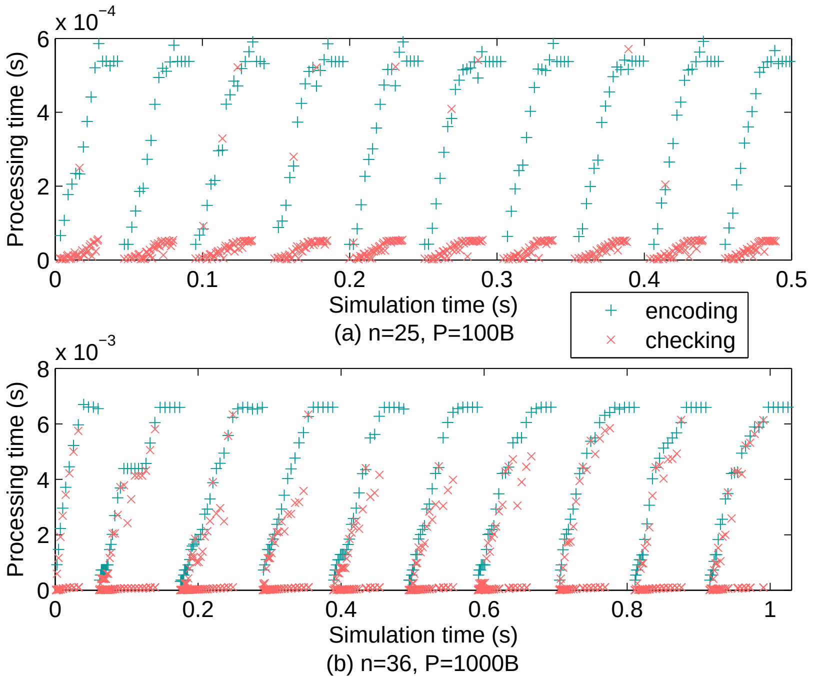


Figure 2.7: Distribution of the processing time of encoding and checking routines.

delay on the transmission and reception of innovative packets. On the other hand, in Fig. 2.7(b), the execution time of the encoding routine can easily become longer than the time interval 850μs. Although such delay severely affects the performance, it still has less time to finish the broadcast than other cases with lower bandwidths. This observation suggests that higher bandwidth is still encouraged in this broadcast algorithm in spite of the more severe impact of processing.

To validate our hardware modeling, we further compare the simulated execution time with the real execution time on the same MIPS processor we have modeled. We measure the execution time of encoding routine on 25 packets with different packet lengths. Fig. 2.8 shows the time difference between the real execution time and the simulated execution time, normalized to the real execution time. For fairness, the time difference is the average of 50 individual comparisons, each with a different random seed. One can see that our simulation provides very high accuracy, over 90% in all cases. Moreover, the simulation has a better accuracy with larger packet length. Since packets are stored and processed sequentially, longer packets can better enjoy the spatial locality, leading to higher cache hit ratio. Thus the simulation without cache behavior yields to results closer to the real execution time.

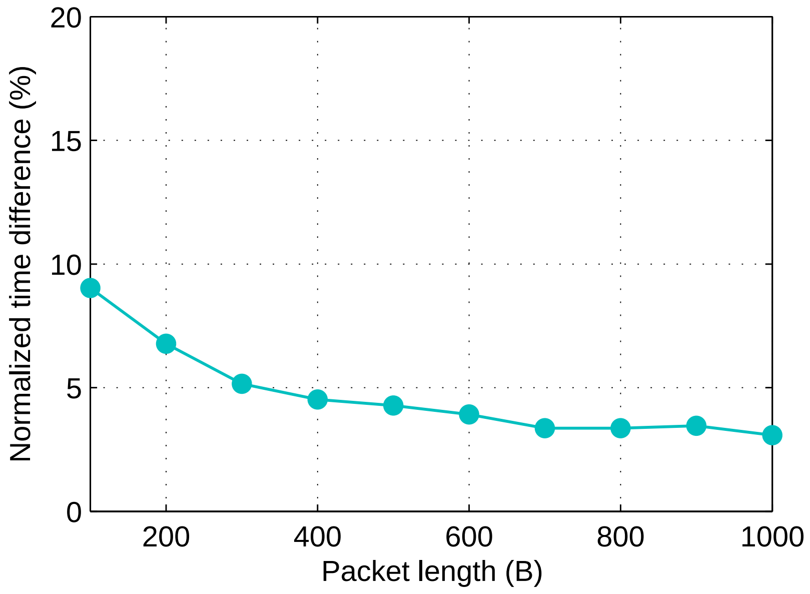


Figure 2.8: Time difference between the real execution and simulation, normalized to the real execution time.

2.5.5 Impact of Task Scheduling

The observation of the interaction between two routines in previous experiments enables us to examine the impact of task scheduling on the broadcasting performance. Since the round-robin scheduling is not suitable as the execution time of either routine is too short to be broken into multiple execution slices, we only consider preemptive scheduling here.

The preemptive scheduling always allows the routine with the highest priority to run. In this experiment, we fix the priority of the encoding routine to 2 and change the priority of the checking routine from 1 to 3 to provide different priority strategies. Note that when the priority is set to 2, the scheduling becomes non-preemptive, which is the scheduling we adopted in previous experiments. When we set the priority of the checking routine to 1, we can see that the broadcasting cannot be completed. In this case, the encoding routine with higher priority will always occupy the CPU, preventing the checking routine from receiving innovative packets. Therefore, we set the priority of the checking routine to 3 in the preemptive scheduling in this experiment.

Table 2.3 provides the performance comparison under preemptive scheduling and non-preemptive scheduling, in which a negative value corresponds to an increased finish time by adopting the preemptive scheduling. We can see that the preemptive scheduling outperforms non-preemptive scheduling with a smaller generation size and a higher bandwidth. A smaller generation size means relatively short execution time of the checking routine, while a higher bandwidth leads to a shorter time interval of which the checking routine will occupy too much fraction. In this case, letting the checking routine run first speeds up the reception of innovative packets without substantial increase in the execution time of the encoding routine, thus reducing the finish time. Otherwise, two schedulings yield very similar results.

Table 2.3: Performance comparison between preemptive and non-preemptive scheduling (where the value is the reduction on finish time under preemptive scheduling divided by finish time under non-preemptive scheduling, and a negative value means preemptive scheduling increases the finish time)

	$B = 54Mb/s$	$B = 11Mb/s$	$B = 2Mb/s$
$n = 25, P = 100B$	13.9%	8.2%	-0.9%
$n = 25, P = 1000B$	3.5%	9.3%	-0.3%
$n = 36, P = 100B$	7.5%	0.6%	-0.7%
$n = 36, P = 1000B$	-0.4%	-0.2%	-0.3%

2.5.6 Impact of Accurate Modeling

To better show the necessity and impact of accurate modeling with co-simulation , we compare the co-simulation result with the result when the experiment is performed in pure NS-2 with the total processing delay largely estimated. In this simulation, we estimate the processing delay based on the time complexity of the routines. Thus, the processing delays of checking and encoding routines are calculated as $c_1 \cdot r \cdot n$ and $c_2 \cdot r(n + P)$, where r is the rank of the current coding matrix. We then use the processing delay obtained in SystemC to calculate the constants c_1 and c_2, each being the average of 500 samples. With these calculations, the estimated processing delay is actually a close approximation of the processing delay in SystemC. For fair comparison, we also design an os_module in NS-2 to implement the non-preemptive scheduling.

Fig. 2.9 presents the results with estimated processing delays normalized to the co-simulation results. To further show the impact of deviated delay estimation, we also perform experiments when the simulated delay is either 0.8 or 1.2 times of the delay estimated above. One can see that when $n = 25$ and $P = 100B$, the results with estimated delays are very close to the co-simulation results. However, when n and P increases, the difference becomes more obvious, especially with higher bandwidth. When $n = 36$, $P = 1000B$ and $B = 54Mb/s$, the finish time with estimated delays is about 30% larger than the co-simulation result. In addition, through the results with deviated delay estimation, we also see that if the processing is not correctly estimated, which is common without the assistance of the real hardware or the SystemC model, the result difference becomes much larger. Such obvious difference indicates that the accurate modeling through co-simulation is required to obtain the accurate results, especially when the actual processing is complicated.

We further compare the detailed behavior of individual nodes during the broadcasting with both simulation approaches. Fig. 2.10 shows the decoding time of the first generation at two nodes under co-simulation and NS-2 with estimated delays when $n = 25$, $P = 100B$ and $B = 54Mb/s$. The decoding time is defined as the time instant when the node receives n innovative coded packets and can decode the current generation of packets. The experiment is repeated 10 times with different seeds for random coding coefficient generation. The clear difference one can observe in Fig. 2.10 is that results under NS-2 simulation are rather stable while results under co-simulation have large variations. On the contrary, the two simulation approaches yield very similar finish time under the same configurations. Such contrast indicates that in spite of the close results sometimes obtained in two simulation approaches on certain performance metric, the detailed behavior can still have large discrepancies. Therefore, to examine the detailed network/hardware behavior, the

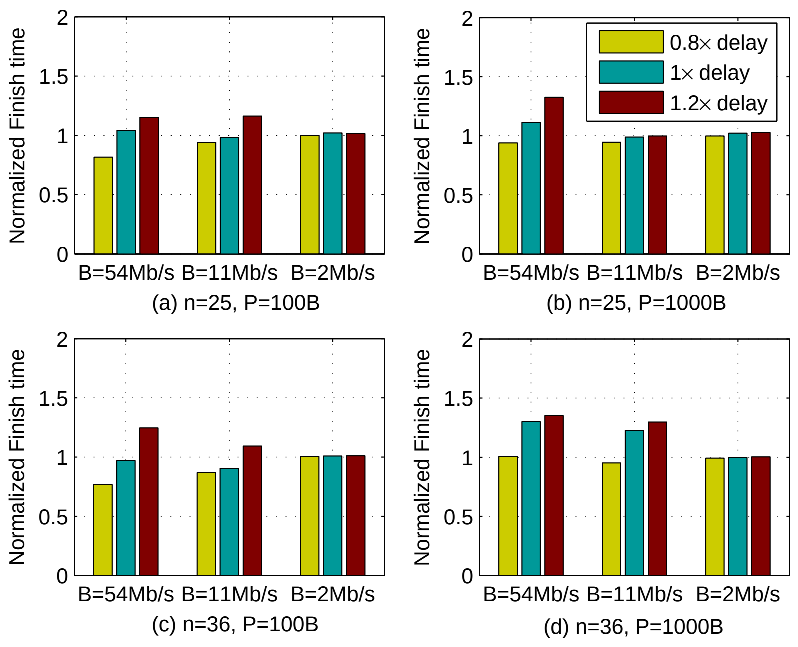


Figure 2.9: The broadcasting finish time with the processing delays estimated in NS-2 where n represents the number of nodes or the generation size, P is the packet length and B is the channel bandwidth. The results are normalized to the corresponding results in co-simulation.

co-simulation, which models processing delays accurately, is irreplaceable.

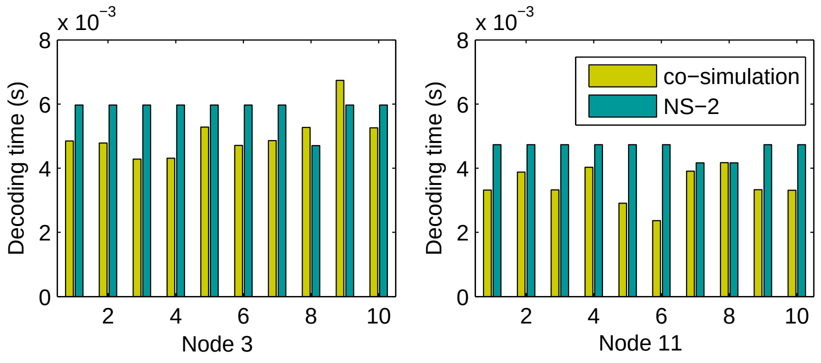


Figure 2.10: The decoding time of the first generation at node 3 and node 11 under co-simulation and NS-2 simulation with estimated delays. The network is configured as $n = 25$, $P = 100B$ and $B = 54Mb/s$. Node 3 and node 11 are at the coordinates (0,3) and (2,1) in the grid, respectively.

2.5.7 Evaluation Summary and Potential Optimization

From the above evaluation results, we can see that the processing of network coding in low-end routers has a non-negligible impact on the overall performance of the broadcasting algorithm. Such impact grows with the increase of channel bandwidth, but there is no linear correlation between them. The actual impact is determined by a combination of the network algorithm, hardware architecture, OS scheduling policy and network configurations, and thus can only be examined on a hardware-aware platform. Finally, despite of such impact, higher network bandwidth should be used as it is likely to yield better overall performance.

Based on the evaluation, we can also perform the following potential optimizations.

- From the perspective of network, the transmission scheduling can be further examined to accommodate the existence of hardware processing delay.
- From the perspective of OS, variable priority could be adopted to further improve the performance.
- From the perspective of hardware, we may choose processors with higher processing capacity or multi-processors to reduce the processing delay.

We may also use specific hardware such as FPGA to speedup the processing.

- All these optimizations can be evaluated on the proposed platform before proceeding to the real implementation, shortening the overall design cycle.

2.5.8 Execution Time of Co-Simulation

As we provide the capability of co-simulation to simulate more accurate network scenarios, the speed of the co-simulation is a concern, which is jointly determined by the simulation speeds of both NS-2 and SystemC. In general, hardware simulation is considered much slower than network simulation, posing a challenge on our co-simulation scheme: If SystemC simulation is performed at a much slower speed, co-simulation will not be able to support experiments for large scale networks that can be originally simulated in NS-2. In this subsection we analyze the execution time of co-simulation to better understand the capability of the platform in terms of simulation speed.

As aforementioned, the co-simulation is performed in two concurrent processes, each corresponding to the execution of a simulator. According to the scheduling algorithm described in Section 2.3, at any time instant, when one process runs the simulation, the other is actually waiting for messages from its counterpart. Thus, the co-simulation logically runs in a sequential way with two processes performing the actual simulation and the communication alternatively. We define a *simulation slice* as the continuous time period in which a process performs simulation without message exchanging. Similarly, we also call each communication as a simulation slice in communication. The execution time of co-simulation can then be divided into a sequence of simulation slices.

We measure the execution time of each simulation slice in our network coding example by a system function in the program, whose resolution is at μs. In the experiment, the packet length is set to $1000B$ and the generation size is set to 36. The computer used in the experiment is equipped with two 64-bit dual-core Intel Xeon processors at 3.8GHz, 8G DDR-2 400 SDRAM and 2M L2 Cache.

Table 2.4 summarizes the statistics of the execution time measured for each simulation slice in both simulators and in communication. The total execution time is about $54.95s$. We can see that the execution time of SystemC is about 10 times as that of NS-2, which is partly due to that the network coding algorithm is implemented in SystemC. To reveal the actual simulation slowdown with co-simulation, we also obtain the execution time when the algorithm is implemented only in NS-2, which is $23.8s$ under the same network configurations. In comparison, the co-simulation increases less than 2 times of the execution time, indicating the hardware simulation speed is comparable with the network simulation speed. This is because our hardware modeling at transaction level accurately estimates the processing time without involving too much detailed hardware operations. Therefore, we believe that appropriate hardware modeling can effectively keep the simulation slowdown within an acceptable degree in the co-simulation.

We also observe that each simulation slice occupies a very small portion in the total execution time, which indicates frequent inter-process communications during the co-simulation. These communications may cause large overhead if not efficiently implemented. As in the table, with the message queue mechanism, the total communication time is within 1% of the total execution time, showing the efficiency of the implementation.

Table 2.4: Statistics of execution times of simulation slices in both simulators and in communication (where the time unit is μs)

Simulation Slice	Sum	Ave	Max	Min
NS-2	4986091	327	13578	16
SystemC	49562317	3247	41488	16
Communication	404549	13	1352	1

2.6 Conclusions

In this chapter, we have presented an efficient, flexible network platform to support the hardware-aware performance study of network algorithms/protocols. Based on the co-simulation of NS-2 and SystemC, the platform provides a network environment where both network traffic and intra-node processing can be well modeled and configured. We have run a case study of wireless all-to-all broadcasting with network coding on the platform to demonstrate the usage of the platform. From the case study, we observed that hardware processing has a great impact on the performance evaluation of network algorithms, which may severely affect the actual performance obtained. We also showed that the impact from hardware processing can be effectively examined under different network and hardware configurations with the support of the platform. Therefore, we believe such a platform is a very useful tool for studying and developing network algorithms and protocols.

Chapter 3

On-Line Adaptive Compression in Delay Sensitive Wireless Sensor Networks

Lossless compression could be a processing intensive network algorithm for wireless sensors. In this chapter, we first utilize the platform introduced in Chapter 2 to evaluate the actual impact of compression on packet delay during data collection. Based on the evaluation we then design two adaptive compression algorithms to maximize the compression benefit in practical wireless sensor networks.

The organization of this chapter is as follows. Section 3.1 introduce the background and motivation. Section 3.2 introduces the LZW compression algorithm and the approach to measuring its execution time in sensor nodes. Section 3.3 characterizes the compression effect on packet delay under various network configurations. Section 3.4 describes the on-line adaptive algorithm in detail, including the analysis of the queueing model and the algorithm implementation in sensor nodes. Section 3.5 examines the performance of compression under the proposed adaptive algorithm. Section 3.6 discusses the enhancement and evaluates the enhanced adaptive algorithm. Finally, Section 3.7 discusses the related work and Section 3.8 concludes the chapter.

3.1 Introduction

Delay sensitive wireless sensor networks require real-time delivery of sensing data to the data sink. Such networks are widely adopted in various real-time applications including traffic monitoring, hazard detection and battlefield surveillance, where decisions should be made promptly once the emergent events occur. Compared to general WSNs where energy efficiency is the primary design concern, delay sensitive WSNs demand more on minimizing the communication delay during data delivery. Recent study in this area has mainly focused on the algorithm design of efficient routing strategies and data aggregation to reduce such delay and provide real-time delivery guarantees [29–32]. In this chapter, we approach this problem from a different and orthogonal angle by considering the effect of data compression. Compression was initially adopted as an effective approach to saving energy in WSNs. In fact, it can also be used to reduce the communication delay in delay sensitive WSNs.

In WSNs, compression reduces the data amount by exploiting the redundancy resided in sensing data. The reduction can be measured by the *compression ratio*, defined as the original data size divided by the compressed data size. A higher compression ratio indicates larger reduction on the data amount and results in shorter communication delay. Thus, much work in the literature has been endeavored to achieve better compression ratio for sensing data. However, from the implementation perspective, most of the compression algorithms are complex and time-consuming procedures running on sensor nodes which are very resource constrained. As the processing time of compression could not be simply neglected in such nodes, the effect of compression on the total delay during data delivery becomes a tradeoff between the reduced communication delay and the increased processing time. As a result, compression may increase rather than decrease the total delay when the processing time is relatively long. In this chapter, we will first analyze this effect in a typical data gathering scheme in WSNs where each sensor collects data continuously and delivers all the packets to a data sink. Then we will design an on-line adaptive algorithm that performs compression only when compression can actually reduce the total delay to guarantee the network to achieve the shortest total delay under all conditions.

To analyze the effect of compression, we need to first obtain the processing time of compression, which depends on several factors, including the compression algorithm, processor architecture, CPU frequency and the compression data. Among numerous compression algorithms, in this chapter we use the Lempel-Ziv-Welch (LZW) [33] as an example, which is a lossless compression algorithm suitable for sensor nodes. We implement the algorithm on a TI MSP430F5418 microcontroller [36], which is used in the current generation of sensor nodes. Our experiments on typical sensing data reveal that the compression time in such a system is comparable to the transmission time of packets, thus cannot be simply ignored. With our hardware aware network platform, we are able to accurately measure the overall network performance through the co-simulation. Our simulation results show that compression may lead to several times longer overall delay under light traffic loads, while it can significantly reduce the delay under heavy traffic loads and increase maximum throughput.

Since the effect of compression varies heavily with network traffic and hardware configurations, we design an on-line adaptive algorithm that dynamically makes compression decisions to accommodate the changing state of WSNs. In the algorithm, we adopt a queueing model to estimate the queueing behavior of sensors with the assistance of only local information of each sensor node. Based on the queueing model, the algorithm predicts the compression effect on the average packet delay and performs compression only when it can reduce packet delay. By conducting extensive simulations on our experimental platform, we show that the adaptive algorithm can make decisions properly and yield near-optimal performance under various network configurations. We further propose an algorithm enhancement that removes the dependency on the network topology with only a small additional transmission overhead. Simulations show that the enhanced adaptive algorithm can achieve better performance in more practical networks.

3.2 LZW Compression in Sensor Nodes

LZW algorithm has been shown to be a suitable compression algorithm for sensor networks. Compared to other compression algorithms, LZW is relatively simple but yields a good compression ratio for sensing data as shown in [34]. In this section, we briefly introduce the LZW algorithm.

LZW compression is a dictionary based algorithm that replaces strings of characters with single codes in the dictionary. The first 256 codes in the dictionary by default correspond to the standard character set. As shown in Table 3.1, the algorithm sequentially reads in characters and finds the longest string s that can be recognized by the dictionary. Then it encodes s using the corresponding codeword in the dictionary and adds string s+c in the dictionary, where c is the character following string s. This process continues until all characters are encoded. A more detailed description of the LZW algorithm can be found in [33].

We will focus on the compression process, as the decompression process is performed at the sink node, which is more powerful thus can perform the decompression in relatively short time. To adapt the LZW compression to sensor nodes, we set the dictionary size to 512, which has been shown to yield good compression ratios in real-world deployments [34].

To achieve a good compression ratio, which is the ratio of the original data

Table 3.1: LZW compression algorithm

```
STRING = get first character
while there are still input characters
    C = get next character
    look up STRING+C in the dictionary
    if STRING+C is in the dictionary
        STRING = STRING+C
    else
        output the code for STRING
        add STRING+C to the dictionary
        STRING = C
    end if
end while
output the code for STRING
```

size to the compressed data size, the string should be long enough to provide sufficient redundancies. Thus, the LZW algorithm is more suitable for WSNs that collect heavier load data, such as images and audio clips. Even in the compression algorithms specifically designed for this type of data, the processing in the algorithms could be complex and time-consuming. Hence, our evaluation result on the LZW algorithm can also provide guidance for adopting these algorithms. In addition, in large scale WSNs, distant nodes require multiple hops of transmissions to reach the sink and the nodes closer to the sink may endure unaffordable traffic. In this case, aggregation is often used (e.g., in cluster-based networks) to reduce the traffic and such aggregated packets consisting of several lighter load packets are also suitable for compression.

3.3 Experimental Study of Compression Effect on Packet Delay

In this section, we study the compression effect on packet delay for data gathering in WSNs. We consider the all-to-one data gathering scenario where all sensors continuously generate packets and deliver them to a single sink. The performance metric used is the end-to-end packet delay, which is the interval from the time a packet is generated at the source to the time the packet is delivered to the sink. To evaluate the compression effect, we compare the end-to-end packet delay with compression and without compression. In the rest of the chapter, we refer to these two schemes as *compression scheme* and *no-compression scheme*, respectively.

3.3.1 Experimental Setup

We examine the compression process on a TI MSP430F5418 microcontroller, which is used in the current generation of sensor nodes. It is a 16-Bit Ultra-Low-Power MCU with 128KB Flash and 16KB RAM. The CPU has a peak working frequency of 18MHz, a very high frequency among the current generation of sensor nodes.

The experiments are conducted in our implemented platform. To simplify the evaluation, we examine the performance on a 2D grid wireless network. Two networks of a 5×5 grid and a 7×7 grid with the sink at the center of each grid are considered. In the simulation, the transmission range is set to $16m$ and the distance between neighboring nodes is $10m$ and $15m$, respectively, to create different network topologies. The packet generation on each sensor node follows an i.i.d. Poisson process, and we assume a fixed packet length for all the packets generated in a single experiment. Two different packet lengths ($256B$ and $512B$) are used to create different compression ratios and processing delays.

At the network layer, we adopt the multi-path routing strategy proposed in [37]. Specifically, each sensor is assigned a level number, which indicates the minimum number of hops required to deliver a packet from this sensor to the sink. Such information can be obtained at the initial setup in the sensor deployment. A sensor with a level number i is called a level i node. A level i node only forwards the packets to its level $i-1$ neighbors. Such a routing strategy is easy to implement, though may not necessarily yield the best real-time performance. However, since no inter-packet compression is involved in our compression strategy, the choice of the routing strategy will not substantially affect the performance evaluation that aims at the compression algorithm.

As we consider delay sensitive networks where packet delay rather than energy efficiency is the primary concern, we adopt the commonly used 802.11 protocol as the MAC layer protocol, and the wireless bandwidth is set to 1Mb/s. The data set is automatically generated by the tool described in [38], which provides a good approximation on the real sensing data in the evaluation of several representative network applications. We use such synthetic data so that our simulation can be performed sufficiently long to capture the steady state behavior of the network without exhausting the simulation data.

Compression is performed on each packet when it is generated at the source node. Since each sensor is equipped with a sequential processor, multiple packets are served in the First-Come-First-Serve order and a sufficiently large buffer is assumed so that no packets are dropped at the compression stage. The compression process is simulated according to the estimation approach described in Section 3.2. The CPU frequency is 18MHz. With these settings, we obtain that the average compression ratio is 1.25 and 1.6 when the packet length is $256B$ and $512B$, respectively. The average processing delay is $0.016s$ for $256B$ packets and $0.045s$ for $512B$ packets. The delays are further compared with the measurements on the real hardware and the simulation accuracy is over 95%. Besides, the processing delay could be increased if lower CPU frequencies or more complex compression algorithms are adopted. For example, it is suggested that the Burrows-Wheeler Transform (BWT) [54] can be performed on the sensing data to assist the subsequent LZW compression to obtain better compression ratio. Such processing delay can greatly affect the performance of high resolution mission-critical applications, including real-time multimedia surveillance and other applications with timeliness requirement, where the extra delay could be critical for some packets to meet their deadlines.

3.3.2 Experimental Results

End-to-End Packet Delay

Based on the routing strategy, a packet generated at a level i node requires i-hop transmissions to reach the sink, resulting in different packet delays for nodes at different levels. In this subsection, we examine the average packet delay for all the nodes at the same level. Fig. 3.1 shows the average delays of different levels in the 5×5 network with neighboring distance set to $15m$ and the packet length set to $512B$.

The primary observation drawn from the figure is that compression has a two-sided effect on the real-time performance depending on the packet generation rate. When the rate is low, compression clearly increases the average delay at each level. For example, when the rate is 2, the average delay of level 1 is increased by about 2.7 times from $13.6ms$ to $51.5ms$ when compression is adopted. Such increase is also observed for other levels, the least of which is 75% for level 4. Note that under such light traffic load, the delay is almost the packet transmission time due to little contention for the wireless channel. Since the packet transmission time reduced by compression is much less than the increase caused by the compression processing time, the overall delay increases, indicating a negative effect of compression. We also notice that such increase is smaller for nodes at higher levels, which can be explained by the fact that nodes at higher levels require more hops of transmissions to reach the sink while each transmission is shortened due to compression. Hence, their

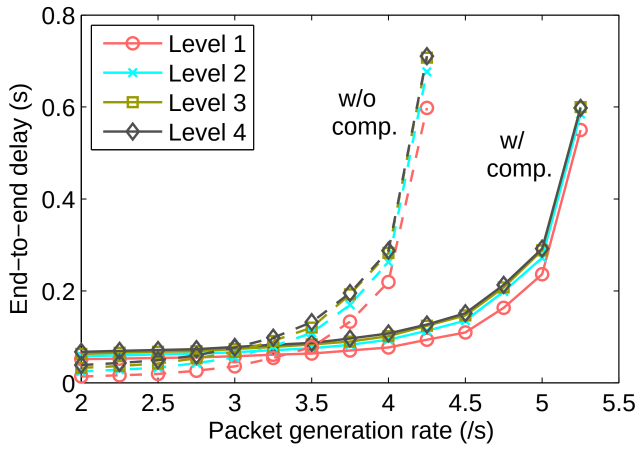


Figure 3.1: Average packet delays for packets generated at different levels.

delay increase caused by compression processing becomes a smaller portion in the total packet delay.

On the other hand, when the packet generation rate becomes higher, the average delays in both cases increase and the increase in the no-compression case grows much faster than that in the compression case. When the rate is higher than 3.5, the compression effect becomes positive and yields significant reduction on average delay. This can be explained as follows. If we consider each node in the network as a queue, the packet generation rate and the packet transmission time are the arrival rate and the service time of the queueing system, respectively. When the traffic is heavy, the transmission time grows rapidly due to channel contentions. Therefore, the utilization of the queueing server, the product of the arrival rate and service time, also grows rapidly, which eventually causes great increase in the average waiting time and the average packet delay. On the other hand, compression shortens the packet length and the transmission time, thus effectively reducing the utilization and the packet delay, which explains the much slower growth of the packet delay with compression.

Another observation we can draw is that the delays of different levels are quite similar. It implies that the main effect of compression is on the transmissions from level 1 nodes to the sink. This is due to the fact that level 1 nodes undergo the heaviest traffic and hence the longest transmission delay. In this case, compression can achieve much more benefit for the transmissions of level 1 nodes than those nodes at higher levels. Thus, based on this observation, it is reasonable to use the average delay among all the nodes in the network to approximate the delays of nodes at different levels. Fig. 3.2 thus shows the average end-to-end delay with different network configurations. While the result reveals a similar trend on the end-to-end delay, we will explore more deeply in the next two subsections to examine the details of the compression effect.

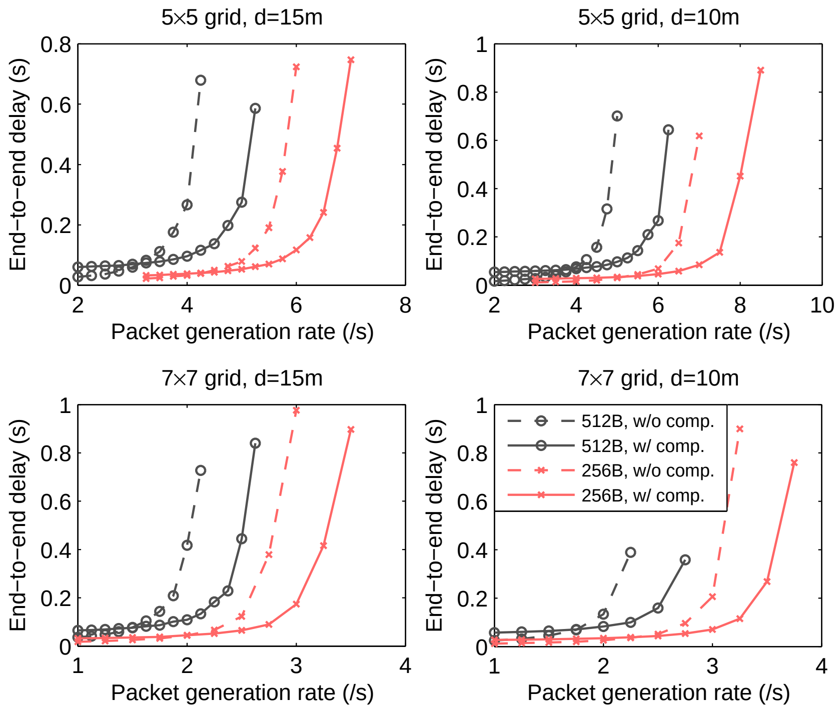


Figure 3.2: Average packet delays under various configurations, where d represents the neighboring distance.

Maximum Packet Generation Rate

In this subsection, we examine the maximum packet generation rate allowed at each node. In Fig. 3.1, we observe that the packet delay grows rapidly when the packet generation rate is relatively high. For example, the average delay without compression reaches $0.7s$ when the generation rate is 4.25, while the delay with compression is $0.6s$ when the generation rate is 5.25. This corresponds to the situation when the utilization of the queueing server approaches 1. To guarantee the success of transmissions, the utilization should be kept below 1. Thus, the generation rate when the utilization approaches 1 is the maximum generation rate allowed in the network. Fig. 3.3 shows the maximum generation rate under different network configurations. Clearly, compression increases the maximum generation rate under all configurations.

In particular, although the maximum generation rate in the no-compression case varies dramatically under different configurations, the relative increase generated by compression remains similar, about 20% to 25%. A small difference is observed for different packet lengths, for example, the increase is 5% lower for the packet length of $256B$ than that for the packet length of $512B$ in the same network configuration. Since the average compression ratio is smaller when the packet length is $256B$, this result indicates that a higher compression ratio leads to a higher maximum generation rate.

Threshold Rate

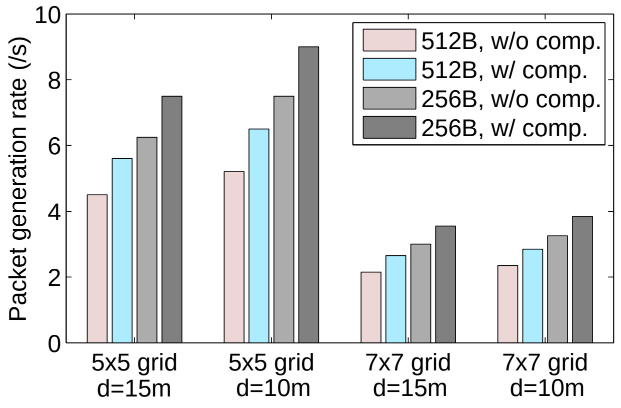


Figure 3.3: Maximum generation rates under various configurations, where d represents the neighboring distance.

Since compression may have either a positive or a negative effect on the packet delay, it would be interesting to find the generation rate, at which the packet delay remains unchanged in both no-compression and compression cases. We call this rate the *threshold rate*. Fig. 3.4 shows the relationship between the threshold rates and the maximum generation rates in the compression case under different network configurations. When the packet generation rate is between the threshold rate and the maximum generation rate, compression can improve the end-to-end packet delay. We can see that the length of this range does not vary much under different configurations, though the threshold rate itself exhibits great variations.

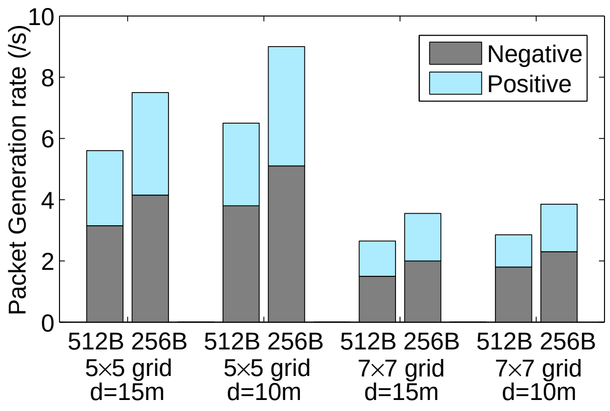


Figure 3.4: Different threshold rates under various configurations, where d represents the neighboring distance.

Summary of the Experimental Study

The above experimental results demonstrate that the delay caused by compression processing is clearly a non-negligible factor in end-to-end packet delay for current generation sensor nodes. Such delay can cause severe performance degradation under light traffic load. On the other hand, when the traffic load is heavier, compression can effectively reduce packet delay and increase maximum throughput. Thus, compression is preferred only when the packet generation rate is higher than the threshold rate. However, the threshold rate varies with network configurations and traffic and thus cannot be obtained in advance. Therefore, it is necessary to design an on-line adaptive algorithm to determine when to perform compression on incoming packets at each node.

3.4 On-Line Adaptive Compression Algorithm

In this section, we present an on-line adaptive compression algorithm that can be easily implemented in sensor nodes to assist the original LZW compression algorithm. The goal is to accurately predict the difference of average end-to-end delay with and without compression by analyzing the local information at a sensor node and make right decisions on whether to perform packet compression at the node. The adaptive algorithm is distributively implemented on each sensor node as *Adaptive Compression Service (ACS)* in an individual layer created in the network stack to minimize the modification of existing network layers. Next we first introduce the architecture of ACS and then describe the algorithm in detail.

3.4.1 Architecture of ACS

The architecture of ACS is described in Fig. 3.5. Located between the MAC layer and its upper layer, ACS consists of four functional units: a controller, an LZW compressor, an information collector and a packet buffer. The controller manages the traffic flow and makes compression decisions on each incoming packet in this layer. The LZW compressor is the functional unit that performs the actual packet compression by the LZW algorithm. The information collector is responsible for collecting local statistics information about the current network and hardware conditions. The packet buffer is used to temporarily store the packets to be compressed.

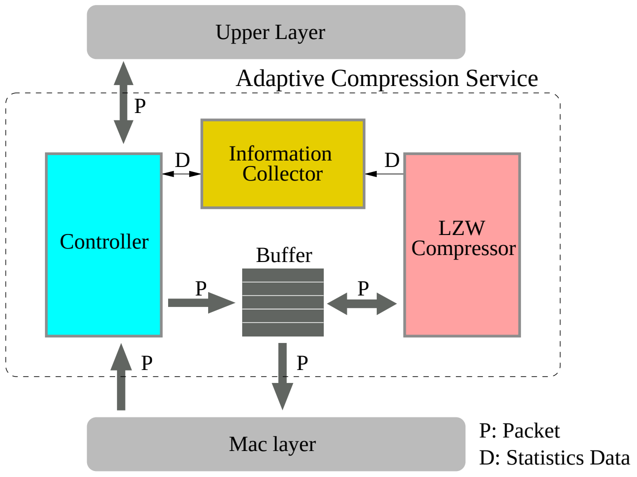


Figure 3.5: Architecture of ACS, which resides in a created layer between the MAC layer and its upper layer.

With ACS, the traffic between the MAC layer and the upper layer is now intervened by the controller in ACS. All outgoing packets coming down from the upper layer are received by the controller, which maintains two states. In the No-Compression state, all packets are directed to the MAC layer without further processing; in the Compression state, only compressed packets, which are received from other nodes, will be directly sent down to the MAC layer, and other packets are sent to the packet buffer for compression. On the other hand, for incoming packets from the MAC layer, only the arrival time is recorded by the collector and the packets themselves are sent to the network layer without delays.

Since compression is managed by the node state, the function of the adaptive algorithm is to determine the node state according to the network and hardware conditions. In our adaptive algorithm, we utilize a queueing model

Table 3.2: Notations

N	Number of sensor nodes in the WSN
R	Radius of the circular region, also the number of levels
n_i	Number of nodes in level i
λ_g	Packet generation rate
λ_e	Arrival rate of external packets from neighbors
p_{kj}	Transition probability, the probability that a packet is served by node k and transferred to node j
λ^i	Mean arrival rate for nodes in level i
T_{tran}	The minimum time of a successful transmission
L_{data}	Length of the data packet
L_{ctl}	Sum of length of all control packets in a transmission
n_{sus}	Number of suspensions in the backoff stage
T_{mac}	MAC layer service time
c_{mac}	Coefficient of variance (COV) of the MAC layer service time
$\overline{T}$	Average packet waiting time of the queue
λ	Arrival rate of the queue
ρ	Utilization of the queue
c_A	COV of the interarrival time
c_B	COV of the service time
r_c	Average compression ratio
T_p	Average compression processing time
c_p	COV of the processing time
p_c	Ratio of compressed packets in all packets
T_{com}	Average packet waiting time at the compression queue
$\Delta T_{mac}(i)$	Delay reduction in level i
ΔT_{min}	Lower bound of total delay reduction

to estimate current conditions based on only local information of sensor nodes. In the next section, we introduce the queueing model.

3.4.2 Queueing Model for a WSN

The queueing model for a WSN includes both the network model and the MAC model, which defines the network topology, traffic model and MAC layer protocol. For a clear presentation, we list the notations that will be used in the model in Table 3.2.

Network Model: Topology and Traffic

We consider a wireless sensor network where N sensor nodes are randomly distributed in a finite two-dimensional region. For calculation convenience, we consider a region of a circular shape with radius R (As will be seen in the experimental results, when the region shape changes, it will not substantially affect the performance of the proposed algorithm). Every node has an equal transmission range r. A data sink is located at the center of the circular region and all sensor nodes send the collected data to the sink via the aforementioned multi-path routing strategy.

If the node density is sufficiently high, we can assume that each node can always find a neighbor whose distance to the sink is shorter than its own distance to the sink by r. Thus, nodes between two circles with radius $(i-1)\cdot r$ and $i \cdot r$ can deliver packets to the sink with i transmissions. According to the routing strategy, these nodes are considered as level i nodes. Without loss of generality, we assume $r = 1$ and there are a total of R levels of nodes. Denote the number of nodes at level i as n_i. Then the average number of nodes at level i can be calculated by

$$E[n_i] = \frac{\pi i^2 - \pi(i-1)^2}{\pi R^2} N = \frac{(2i-1)N}{R^2} \tag{3.1}$$

Such a network can then be represented by an open queueing network where each sensor node is modeled as a queue with an external arrival rate λ_g, which corresponds to the packet generation rate. Denote λ_j^i as the arrival rate of node j at level i. When $i < R$, by queueing theory, we have

$$\lambda_j^i = \lambda_g + \sum_{k=1}^{n_{i+1}} \lambda_k^{i+1} p_{kj} \tag{3.2}$$

where p_{kj} is the transition probability from node k at level $i+1$ to node j at level i.

As all λ_j^i's have the same expected value, we denote it as λ^i. Summing λ_j^i and taking expectation on both sides of the equation lead to

$$E[n_i]\lambda^i = E[n_i]\lambda_g + \lambda^{i+1} E\left[\sum_{j=1}^{n_i} \sum_{k=1}^{n_{i+1}} p_{kj}\right] \tag{3.3}$$

Since $\sum_{j=1}^{n_i} \sum_{k=1}^{n_{i+1}} p_{kj} = n_{i+1}$, the above equation can be written as

$$\lambda^i = \lambda_g + E\left[\frac{n_{i+1}}{n_i}\right] \lambda^{i+1} = \lambda_g + \frac{2i+1}{2i-1}\lambda^{i+1} \tag{3.4}$$

Thus, each node can estimate the arrival rates of nodes at other levels based on their own arrival rates. To evaluate the performance of the queueing system, we also need another important parameter, the average packet service time, which includes the possible packet compression time and the MAC layer service time. While the compression time can be obtained directly from the LZW algorithm, the calculation of the MAC layer service time requires an understanding of the MAC model.

MAC Model

The MAC layer packet service time is measured from the time when the packet enters the MAC layer to the time when the packet is successfully transmitted or discarded due to transmission failure. To analyze the packet service time, we first briefly describe the Distributed Coordination Function (DCF) of IEEE 802.11 [55] used as our MAC layer protocol.

DCF employs a backoff mechanism to avoid potential contentions for the wireless channel. To transmit a packet, a node must conduct a backoff procedure by starting the backoff timer with a count-down time interval, which is randomly selected between $[0, CW)$ where CW is the contention window size. The timer is decremented by 1 in each time slot when the channel is idle and is suspended upon the sensing of an ongoing transmission. The suspension will continue until the channel becomes idle again. When the timer reaches zero, the node completes the backoff procedure and starts transmission. The entire procedure is completed if the transmission is successfully acknowledged by the receiver. Otherwise, the transmission is considered failed, which invokes a retransmission by restarting the backoff timer. In each retransmission, the contention window size CW will be doubled until it reaches the upper bound defined in DCF. Finally, the packet will be discarded if the number of retransmissions reaches the predefined limit.

In our MAC layer, we also adopt the RTS/CTS mechanism to reduce transmission collisions. Thus, it requires at least 4 transmissions to successfully transmit a data packet: the transmissions of RTS, CTS, data and ACK packets. Let T_{tran} denote the minimum packet transmission time, which can be calculated as the sum of all packet lengths divided by network bandwidth Bw

$$T_{tran} = \frac{L_{data} + L_{ctl}}{Bw}, \tag{3.5}$$

where $L_{ctl} = L_{RTS} + L_{CTS} + L_{hdr} + L_{ACK}$ and L_{hdr} is the header size of the data packet. Let T_{sus} denote the average duration of the timer suspension in the backoff stage and T_{col} denote the average time spent in transmission collisions. The suspension duration is actually the time waiting for other nodes to complete a packet transmission. Under the assumption of the constant packet length, we can approximately have $T_{sus} = T_{tran}$. On the other hand, with the RTS/CTS mechanism, the collision mainly occurs during the transmission of RTS and continues until the timeout for CTS. Hence $T_{col} \approx \frac{L_{RTS}+L_{CTS}}{Bw}$. The overall MAC layer packet service time can then be calculated as

$$\begin{aligned} T_{mac} &= n_{sus}T_{sus} + n_{col}T_{col} + T_{tran} \\ &= (n_{sus}+1)T_{tran} + n_{col}T_{col} \\ &\approx (n_{sus}+1)T_{tran} \end{aligned} \tag{3.6}$$

where n_{sus} represents the number of suspensions and n_{col} is the number of transmission collisions. The last step approximation in Equation (3.6) is due to the relatively small value of both n_{col} and T_{col}. We also exclude the backoff time and some interframe spaces in the above equation due to the same reason.

Queueing Analysis for a Sensor Node

The queueing model of a sensor node is different in different node states. In the No-Compression state, each node is considered as a single queue. Its arrival process is a combination of the local packet generation process and the departure processes of its neighbors that send packets to the node. As the simulation results in [60] showed, the departure process of nodes adopting IEEE 802.11 MAC protocol can be approximated as a Poisson process. Thus, we assume that the arrival process of each node is a Poisson process and each node is an M/G/1 (i.e., exponential interarrival time distribution/general service time distribution/single server) queue. In the Compression state, the queueing model of each node can be modified as a system of two queues as shown in Fig. 3.6 with the compression queue and the transmission queue corresponding to ACS and the MAC layer, respectively. The compression queue can be modeled as a M/G/1 queue because its arrival process, as a split of the arrival process of the sensor node, can also be considered as a Poisson process. On the other hand, since its departure process, a part of the arrival process of the transmission queue, is not a Poissonian, we model the transmission queue as a GI/G/1 queue where GI represents the general independent interarrival time distribution.

For M/G/1 queue, according to the well known Pollaczek-Khinchin formula, the average number of packets $\overline{N}$ in a M/G/1 queue can be calculated as

$$\overline{N}_{M/G/1} = \rho + \frac{\rho^2}{1-\rho} \cdot \frac{1+c_B^2}{2}, \tag{3.7}$$

where ρ is the utilization of the queue and c_B is the coefficient of variance

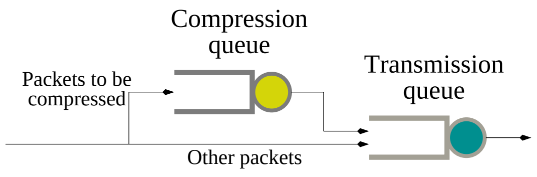


Figure 3.6: Queueing model of a sensor node with compression.

(COV) of the service time. By Little's Law, given the arrival rate λ, the average packet waiting time, which is the packet delay in the node, can be derived as

$$\overline{T}_{M/G/1} = \frac{\overline{N}}{\lambda} = \frac{2\rho - \rho^2(1 - c_B^2)}{2\lambda(1 - \rho)} \tag{3.8}$$

To derive the average packet waiting time for the transmission queue, we use the diffusion approximation method [64], which is briefly introduced here. Consider an open GI/G/1 queueing network with n nodes. Given the arrival rate λ_k for node k, $k \in [1, n]$, the COV of the interarrival time at node j, c_{Aj}, can be approximated as

$$c_{Aj}^2 = 1 + \sum_{k=0}^{n} (c_{Bk}^2 - 1) \cdot p_{kj}^2 \cdot \lambda_k \cdot \lambda_j^{-1} \tag{3.9}$$

where c_{Bk} represents the COV of the service time at node k and c_{B0} represents the COV of the external interarrival time. In Fig. 3.6, we assume the COV of the service time at the compression queue is c_p and the portion of compressed packets in all packets is p_c. Then we obtain the COV of the service time at the transmission queue as

$$c_A^2 = 1 + (c_p^2 - 1)(1 - p_c) \tag{3.10}$$

The average packet waiting time can then be approximated by

$$\overline{T}_{GI/G/1} = \frac{\rho}{\lambda(1 - \hat{\rho})} \tag{3.11}$$

where

$$\hat{\rho} = \exp\left(- \frac{2(1 - \rho)}{c_A^2 \cdot \rho + c_B^2} \right) \tag{3.12}$$

3.4.3 Adaptive Compression Algorithm

We are now in the position to describe the adaptive compression algorithm, which can be divided into two stages: information collection and node state determination.

Information Collection

In ACS, the information collector is responsible for collecting three types of statistics information:

1. Compression statistics, including the average compression ratio r_c, the average compression processing time T_p and the COV of the processing time c_p. Upon each packet compression, the compression ratio, the processing time and its square are recorded. The statistics is updated after every m packets are compressed. In our experiments, we set $m = 100$. Since the compression statistics can only be collected when nodes perform compression, the network should include an initial phase when all nodes perform compression to obtain the initial values. We assume the sensing data has a stable distribution so that the collecting method above is sufficient to provide proper statistics.

2. Packet arrival rates, which include the arrival rate of external packets from neighbors λ_e and the packet generation rate λ_g. The calculations follow a time-slotted fashion. In each time slot, the total number of packets arrived is counted and the arrival rates are calculated by dividing the total counts by time. Noting that not all external packets are compressed in the adaptive algorithm, we also record the ratio of compressed packets in the external packets as p_c.

3. MAC layer service time. Since the MAC layer service is considered as an arbitrary process in the queueing model, its mean T_{mac} and COV c_{mac} are calculated and recorded for the subsequent analysis. The calculation is done along with the calculation of arrival rates in the same time slot to reduce the implementation complexity.

State Determination

In ACS, the controller determines the appropriate node state according to the statistics information collected. Since most of information is collected in a time-slotted manner, the decision on the node state is made at the end of each time slot. Depending on the current node state, the decision process is slightly different. Thus we first consider the case when the node is currently in the

No-Compression state. Then the task of the controller is to decide whether performing compression on this node can reduce packet delays. From the experimental results in Section 3.3, we know that compression introduces an extra delay due to compression processing and reduces the packet delay from the current node to the sink. We now discuss these two delays separately.

The incoming packets of the compression queue are the uncompressed portion of the arrival packets at the node. With the information provided by the collector, the arrival rate can be calculated by $\lambda_c = \lambda_e(1-p_c)+\lambda_g$, while the service rate is $\frac{1}{T_p}$, and the utilization equals $\lambda_c T_p$. By Equation (3.8), the increased compression time for a packet can be derived as

$$T_{com} = \frac{1}{2} \cdot \frac{2T_p - \lambda_c T_p^2(1-c_p^2)}{1-\lambda_c T_p}. \tag{3.13}$$

As the ultimate goal of compression is to reduce average delay, which is equivalent to reducing the sum of delays of all packets, we define *normalized delay* as the total delay for all packets in a unit time. Thus, for a time interval t, the total increased delay due to compression is $\lambda_c t T_{com}$, and the normalized delay increase is $\lambda_c T_{com}$.

Now let's look at the delay reduction in packet delivery after compression. While it is difficult to accurately calculate this reduction, we can easily obtain a lower bound. Given the compression ratio r_c and the length of the original packet L, the packet length is shortened by $L - \frac{L}{r_c}$ after compression and its transmission time is reduced by at least $\frac{L(r_c-1)}{r_c \cdot Bw}$. For a level i node, there are i transmissions for this packet after compression. Thus, the total reduction is at least $\Delta T_{min} = \frac{i \cdot L(r_c-1)}{r_c \cdot Bw}$. We compare this lower bound with the increased compression time T_{com}. If $T_{com} \le \Delta T_{min}$, the node is switched to the Compression state.

When $T_{com} > \Delta T_{min}$, we need to calculate the normalized delay reduction. In fact, if only one node performs compression, there will be no extra reduction other than the reduced transmission time because the surrounding traffic will not change. However, according to our network model, nodes at the same level should share very similar traffic conditions, thus although made independently, their state determination is likely to coincide. Therefore, when we consider the delay reduction due to compression on a node, it is reasonable to assume that other nodes at the same level will also perform compression. In this sense, compression actually affects the transmissions of all packets in the node rather than only the uncompressed packets. Furthermore, such effect not only resides in the local node, but also in the downstream nodes on the routing path when they receive those compressed packets. Thus, we need to examine the delay reduction level by level.

We first examine the delay reduction in the local node. Without loss of generality, we assume the node is at level i. The packet delay before compression can be obtained by Equation (3.8), where $\lambda = \lambda_g + \lambda_e$, $\rho = \lambda \cdot T_{mac}$ and $c_B = c_{mac}$. When compression is performed, the transmission queue is modeled as a GI/G/1 queue and the packet delay is calculated by Equations (3.11) and (3.12). According to Fig. 3.6, the arrival rate of the transmission queue does not change after the compression. c_A can be calculated by Equation (4) and c_B is approximated to be c_{mac}. Next, we derive the average service time after compression from T_{mac}. Based on the analysis in Section 3.4.2, the service time T_{mac} is approximately proportional to the packet transmission time T_{tran} and hence $L_{data} + L_{ctl}$. We assume the original packet length is L and the compressed packet length is then L/r_c where r_c is the compression ratio obtained in the collector. Thus L_{data} is reduced from $L - p_c(L - L/r_c)$ to L/r_c by compression while L_{ctl} keeps unchanged. The average service time after compression can then be calculated as

$$\begin{aligned} T'_{mac} &= T_{mac} \cdot \frac{L/r_c + L_{ctl}}{L - p_c(L - L/r_c) + L_{ctl}} \\ &= \frac{T_{mac}(L + L_{ctl} \cdot r_c)}{L(r_c + p_c - r_c \cdot p_c) + L_{ctl} \cdot r_c} \end{aligned} \tag{3.14}$$

With all these parameters known, the packet delay after compression can be obtained and so is the delay reduction, denoted as $\Delta T_{mac}(i)$.

Now we estimate the delay reduction at a level k node ($k < i$) on the routing path. According to the analysis in Section 3.3, higher level nodes enjoy more compression benefit through the transmissions on longer routes, indicating that they tend to perform compression more aggressively. Given the local node in the No-Compression state, it is reasonable to assume that the nodes with the same packet generation rates at level k are also in the No-Compression state. Then the packet delay at this node is calculated with Equation (3.8). By Equation (3.4), we can derive the arrival rate by λ_g and λ_e. The calculation of the service time T_{mac} mainly depends on the value of n_{sus} and T_{tran}. While T_{tran} can be calculated in a similar way to the local node, n_{sus} cannot be derived intuitively. In [62], it was shown that n_{sus} is proportional to the utilization ρ, if ρ is shared by all nodes in its interfering range. For our network, we approximately assume that nodes in the interfering range of a level k node are all in level k. Then we have

$$\begin{aligned} \frac{n^k_{sus}}{n^i_{sus}} &= \frac{\rho^k}{\rho^i} = \frac{\lambda^k}{\lambda^i} \cdot \frac{T^k_{mac}}{T^i_{mac}} = \frac{\lambda^k}{\lambda^i} \cdot \frac{(n^k_{sus}+1)T^k_{tran}}{(n^i_{sus}+1)T^i_{tran}}, \\ n^k_{sus} &= \frac{c \cdot n^i_{sus}}{n^i_{sus} - c \cdot n^i_{sus} + 1}, c = \frac{\lambda^k T^k_{tran}}{\lambda^i T^i_{tran}} \end{aligned} \tag{3.15}$$

where parameters for level i and level k nodes are labeled with the superscript i and k respectively. With n_{sus} and T_{tran}, the service time can be obtained and so is the reduction, which we denote as $\Delta T_{mac}(k)$.

The normalized delay reduction can be calculated as

$$\Delta T_{mac} = \sum_{j=1}^{i} \lambda^j \Delta T_{mac}(j). \tag{3.16}$$

The decision on the state of the node is then made by comparing ΔT_{mac} with $\lambda_c T_{com}$. The procedure of the state determination in pseudo code is given Table 3.3.

When the node is currently in the Compression state, all the calculations above will also be performed. The only difference is that we compare the reduced processing time with the increased transmission time due to compression cancellation.

3.5 Performance Evaluations

With the proposed on-line adaptive compression algorithm, each node can dynamically decide whether a packet is compressed or not, adapting to the current network and hardware environment. In this section, we present the performance evaluation results for the proposed adaptive compression algorithm. We compare the performance of our adaptive scheme with other two schemes: the No-compression scheme in which no packets are compressed at all, and the Compression scheme where all packets are compressed in data gathering.

The experiments are conducted in three networks: a 7×7 grid, a 9×9 grid and a 11×11 grid. The sink is located at the center of the grid. The transmission range is 1.5 times of the neighboring distance. Then the three networks have 3, 4 and 5 levels of nodes, respectively. The original packet length is set to 512B. Other parameters are similar to the configuration in the previous experiments in Section 3.3.

In the simulation, the packet generation rate starts at a rate lower than

Table 3.3: State determination procedure, performed at the end of each time slot for a node in the No-Compression state

```
For each node at level $i$:
**if** state = No-Compression **then**
    read statistics from the information collector
    compute $T_{com}$ and $\Delta T_{min}$
    **if** $T_{com} \leq \Delta T_{min}$ **then**
        set state to Compression
    **else**
        set $i$ to the node's level number
        set $\Delta T_{mac}$ to zero
        **while** $i > 0$
            calculate $\lambda^i$
            compute reduction $\Delta T_{mac}(i)$
            add $\lambda^i \Delta T_{mac}(i)$ to $\Delta T_{mac}$
            decrease $i$ by one
        **end while**
        **if** $\lambda_c T_{com} \leq \Delta T_{mac}$ **then**
            set state to Compression
        **end if**
    **end if**
**end if**
```

the threshold rate and then increases linearly every 300 seconds. Such increase continues until the maximum rate is reached, at which time, the simulation also ends. The time slot length used in the information collector is $30s$. Fig. 3.7 shows the average end-to-end delays for three schemes in different networks. We can see that the delay for the adaptive scheme is always close to the lower one of the No-Compression and Compression schemes, indicating overall good adaptiveness of the algorithm on the network traffic. In particular, when the generation rate is lower than the threshold rate, the adaptive scheme chooses not to compress packets and thus yields nearly the same results as the No-compression scheme. When the generation rate is slightly higher than the threshold, the adaptive scheme yields similar results to the Compression scheme except some points with slightly longer delays in the 9×9 grid and the 11×11 grid. The largest increase in delay is only 10% when the generation rate is 0.9 in the 11×11 grid. Such consistently good performance in different networks indicates a good scalability of the algorithm on the network size.

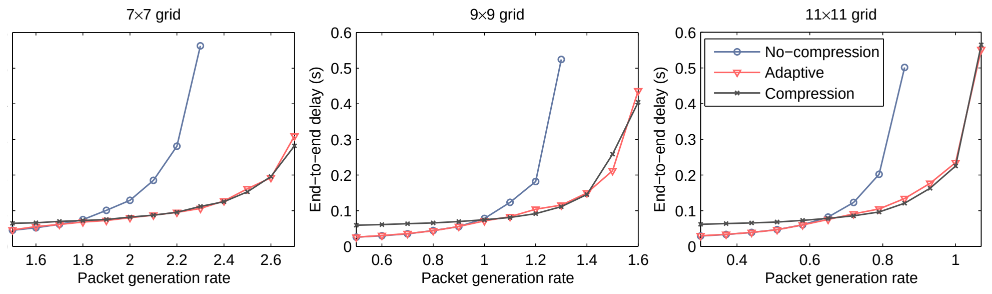


Figure 3.7: The overall average packet delays under three schemes in three different networks.

We further examine the delays for different levels to provide a comprehensive illustration. Fig. 3.8 depicts the average delays for different levels in the 9×9 grid network. It can be noticed that for nodes at level 1, the adaptive scheme outperforms other two schemes when the generation rate exceeds the threshold rate. For nodes at levels 3 and 4 at the same generation rate, however, the average delays under the adaptive scheme are slightly longer than that under the Compression scheme. This can be explained by the observation that the threshold rates for higher level nodes are lower than the thresholds for lower level nodes. When the generation rate is between two different threshold rates, higher level nodes perform compression while the lower level nodes do not. Therefore, without suffering from the compression processing delay, lower level nodes can still enjoy the benefit of the reduced average packet length as they receive packets from higher level nodes, resulting in shorter delays than that under Compression scheme. On the other hand, since lower level nodes do not perform compression, the average packet length and the subsequent transmission delay become longer, increasing the end-to-end delay for other nodes. Such increase is also more evident for higher level nodes as the generated packets in these nodes endure more hops of prolonged transmissions during the delivery to the sink.

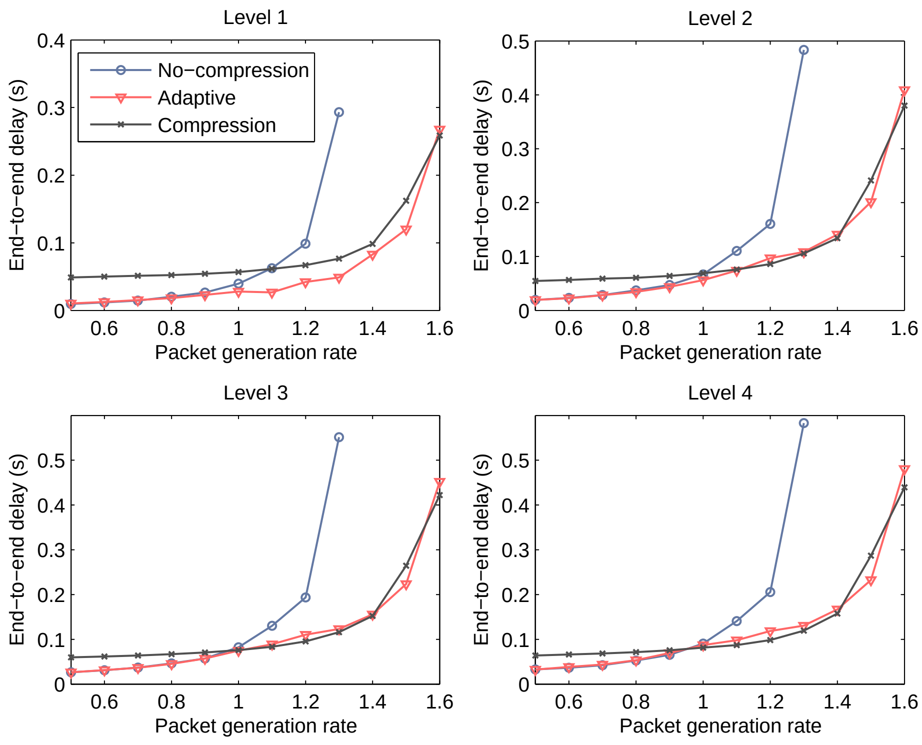


Figure 3.8: The average delays for packets generated at different levels under three schemes in a 9×9 grid network.

3.6 Enhanced Adaptive Algorithm

So far we have shown that the adaptive algorithm can correctly estimate the compression benefit based on only local information and achieve good performance. It should be noticed that such estimation relies on the derivation of the arrival rate and MAC layer service time of other nodes under the assumption that exactly R levels of nodes are distributed in the circular region with radius R. Though valid with sufficiently high node density, this assumption may not always hold in practical scenarios. For example, a node between circle R and $R-1$ may not find an R-hop but an $R+1$-hop route to reach the sink, making itself an $R+1$ level node. In this case, the derivation of arrival rates of other nodes becomes inaccurate for such nodes, deteriorating the algorithm performance. In this section, we introduce an enhanced algorithm to solve this problem.

The adaptive algorithm calculates the arrival rates of other nodes by Equation (3.4), which requires the estimation on the number of nodes in each level, or the ratios of the nodes between two adjacent levels. Let r_i denote the ratio between the number of nodes in levels i and $i+1$. The original estimation $r_i = \frac{2i+1}{2i-1}$ may not be applicable in some practical scenarios. Instead, we could obtain r_i from the arrival rates

$$r_i = \frac{\lambda^i - \lambda_g}{\lambda^{i+1}}, \tag{3.17}$$

where λ^i is the average arrival rate of nodes in level i and λ_g is the packet generation rate. Therefore, if the average arrival rates of all k $(k \leq i)$ levels can be obtained, a level i node can correctly estimate the required ratios and thus the compression benefit.

In the adaptive algorithm, nodes in level i utilize their own arrival rates to represent the average arrival rate of level i. Such approximation is made based on that the compression decision on a node only directly affects its neighbors within a limited range. Therefore, a level i node can obtain all λ^k for $k < i$ by acquiring the arrival rates of its downstream nodes on the route to the sink. Since MAC 802.11 requires acknowledgement for each packet transmission, these arrival rates can be easily included in the ACK packets and passed backward on the route. Such piggybacking minimizes the modification on the MAC layer protocol. Note that this process creates very small communication overhead as the added data is bounded by the number of levels in the network. Moreover, as we assume a static network topology, ratio r_i will stay unchanged so that the extra transmissions can be terminated once every node receives the required arrival rates.

We now evaluate the performance of the enhanced adaptive algorithm in a more practical scenario. In the simulations, 90 and 180 nodes are randomly distributed in a circular region with radius 3 to create two different network topologies which consist of 5 and 4 levels of nodes, respectively. The packet length is set to be $512B$ and $256B$ and other parameters are similar to previous experiments.

Fig. 3.9 shows the average end-to-end delay under both the original and enhanced versions of the adaptive algorithm. To better examine the com-

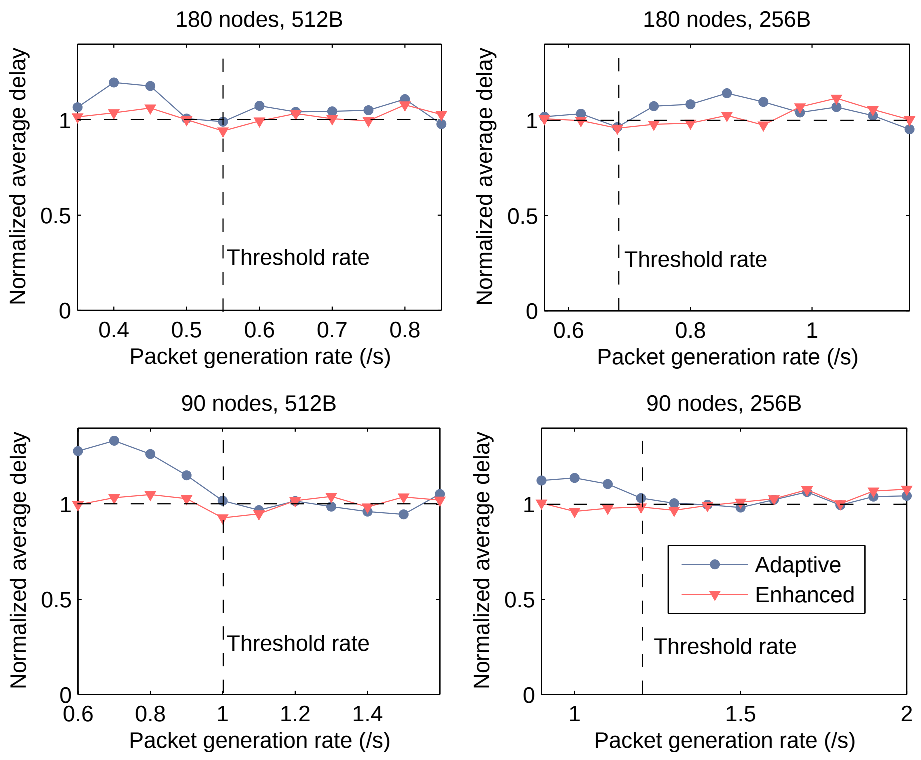


Figure 3.9: Average packet delays under various configurations. The value is normalized to the corresponding values in an idealized scheme where packets are compressed only if the packet generation rate exceeds the threshold rate.

pression performance, we assume an ideal scheme where the threshold rate is obtained beforehand and packets are compressed only if the packet generation rate exceeds the threshold rate. The threshold rate is obtained by comparing the results under Compression and No-compression schemes as in Section 3.3 and is represented by the vertical dashed line in Fig. 3.9. Then the experimental results are normalized to the corresponding delays in the idealized scheme. We observe that the average delay in the enhanced algorithm is approximately equal to the delay in the ideal scheme, with less than 10% difference. On the other hand, the original algorithm creates an obvious increase in packet delay when the packet generation rate is lower than the threshold rate. This can be explained by the inaccurate estimation on r_i, which exaggerates the compression benefit and leads to over-aggressive compression decision. Furthermore, such inaccuracy for sparser node density will become greater, which well explains the larger increase in the 90-node scenario.

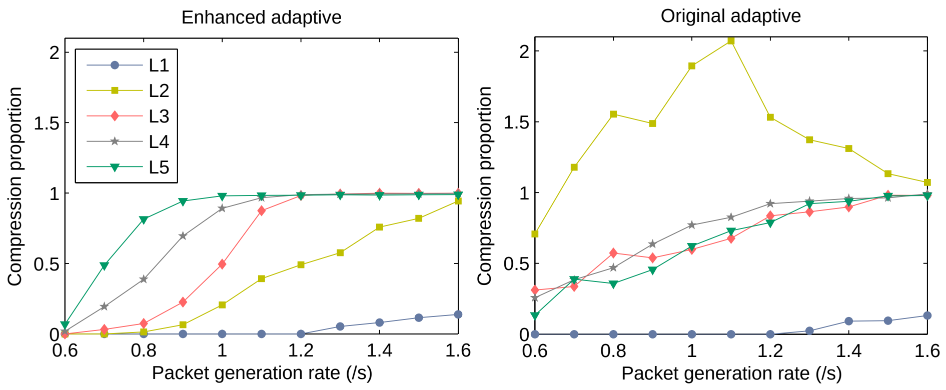


Figure 3.10: The compression proportion of each level. The number of nodes is 90 and the packet length is $512B$.

To better understand the behavior of two adaptive algorithms, we record the *compression proportion* of each level, which is calculated as the number of compressed packets divided by the number of generated packets in each level. Notice that this proportion may be larger than 1 because the packets compressed at nodes in one level could be generated at nodes in other levels. Fig. 3.10 shows the compression proportion of the two algorithms when the number of nodes is 90 and the packet length is $512B$. With the enhanced algorithm, compression is first performed in upper level nodes when the packet generation rate grows, which is consistent with the observation in Section 3.3 that the threshold rates in upper levels are smaller than those in lower levels. When the packet generation rate is sufficiently high, the proportion of compressed packets in most levels goes to 1 and thus most nodes perform compression. By contrast, with the original algorithm, due to the inaccurate estimation, a large amount of packets are compressed when the packet generation rate is much smaller than the threshold rate. Besides, the proportion of compressed packets in level 2 is larger than 1 in most cases, which means that many packets generated in upper level nodes are compressed after transmitted to nodes in level 2. Such behavior actually remedies the false decisions made in upper level nodes, still yielding near-optimal performance when the threshold rate is exceeded as shown in Fig. 3.9. In addition, we observe that level 1 nodes rarely perform compression in both algorithms. However, it does not affect the performance heavily as most packets are already compressed when they are transmitted to nodes in level 1.

3.7 Related Works

In this section, we discuss some existing compression approaches for WSNs and their feasibility to be adopted in our adaptive algorithm. In addition, since the algorithm relies heavily on the performance analysis in IEEE 802.11 based networks, we also briefly review the previous analysis on the IEEE 802.11 MAC protocol.

3.7.1 Compression in WSNs

Due to the limited energy budget in sensor networks, compression is considered as an effective method to reduce energy consumption on communications and has been extensively studied. In the meanwhile, the spatial-temporal correlation in the sensing data makes it suitable for compression. In general, works related to compression in WSNs can be classified into three categories. The first category is the theoretical work. Akyildiz, et al. proposed a theoretical framework to model the correlation and its utilization in several possible ways [39]. Scaglione and Servetto proved that the optimal compression efficiency can be achieved by source coding with proper routing algorithms [40]. The relationship between compression and hierarchical routing was also discussed in [41]. Pattem, et al. analyzed the bound on the compression ratio achieved by lossless compression and showed that a simple, static cluster-based system design can perform as well as sophisticated adaptive schemes for joint routing and compression [42]. In [43], a framework of distributed source coding (DSC) was given to compress correlated data without explicit communications. However, the implementation of DSC in practice could be difficult as it requires the global knowledge of the sensing data correlation structure.

The second category consists of lossy compression algorithms in the sense that the compressed data may not be fully recovered by decompression. A series of algorithms utilized the data correlation to reduce the amount of data to be gathered while keeping the data within an affordable level of distortion. The approaches include data aggregation [45, 46], where only summarized data are required, approximate data representation using polynomial regression [47], line segment approximation [48] and low-complexity video compression [49].

The third category is lossless compression which can reconstruct the original data from the compressed data. Coding by Ordering [50] is a compression strategy that utilizes the order of packets to represent the values in the suppressed packets. PINCO [51] was proposed to compress the sensing data by sharing the common suffix for the data. This approach is more useful for data with long enough common suffices, such as node ID and time stamp. In [34], the implementation of LZW compression algorithm in sensor nodes was extensively studied and showed to have a good compression ratio for different types of sensing data. Another compression algorithm, Squeeze.KOM [52], used a stream-oriented approach that transmits the difference, rather than the data itself to shorten the transmission. In addition, a survey on several compression algorithms was given in [53].

Compared to the previous work, the purpose of this chapter is not to design another compression algorithm. Rather, we propose an adaptive compression framework, which determines when compression should be performed depending on the compression performance and the network condition. Such framework is not restricted to the LZW algorithm adopted in the chapter and is suitable for any source compression approaches.

3.7.2 Performance Analysis of IEEE 802.11

The proposed adaptive algorithm is based on the analysis of IEEE 802.11 DCF, which has been extensively studied in the literature. Bianchi [56] presented a Markov chain model to analyze the throughput performance under saturated traffic condition where nodes always have packets to send. In [57], a modification on this model was made and the analysis was extended to model both throughput and delay performance. Similar analysis under different assumptions can be found in [58, 59]. While these works considered saturated traffic condition, the unsaturated condition was further examined in [60]. Recently, the analysis was extended to consider the performance of multi-hop networks [61–63], which is closely related to the analysis in this chapter. In [61], one hop delay was considered in multi-hop networks where nodes are divided into source nodes and relay nodes. The source node generates packets and only sends packets generated by itself while the relay node does not generate packets and only forwards the received packets. In [62], the end-to-end delay was analyzed in a torus network where each node could be a source, destination or relay node. A packet received by a node is absorbed by the node with the probability of $p(n)$ and forwarded to the neighbors with the probability of $1 - p(n)$. Compared to these simplified traffic models, a similar traffic model to our work was adopted in [63]. An analytical framework based on Markov chain was proposed to accurately predict the distribution of end-to-end delay by considering a series of parameters, including traffic pattern, MAC layer protocols and link quality. However, the calculation in their analysis is quite intensive and requires powerful processors, such as Xeon CPU as originally used in [63]. Therefore, such analysis is unaffordable for sensor nodes. In contrast, our analysis is more focused on understanding the correlation between the packet length and the end-to-end delay, as well as the correlation between delays in different levels. With the analysis, we can easily estimate the packet delay based on local measurements, rather than obtaining the packet delay directly from theoretical analysis.

3.8 Conclusions

In this chapter, we have studied the effect of the compression on end-to-end packet delay in data gathering in WSNs. We accurately examine the effect of compression by measuring the execution time of a lossless compression algorithm LZW on microcontroller TI MSP430F5418. Through extensive simulations, we found that compression has a two-sided effect on packet delay in data gathering. While compression increases the maximum achievable throughput, it tends to increase the packet delay under light traffic loads and reduce the packet delay under heavy traffic loads. We also evaluated the impact of different network settings on the effect of compression, providing a guideline for choosing appropriate compression parameters.

We then proposed an on-line adaptive compression algorithm and an enhancement to make compression decisions based on the current network and hardware conditions. The adaptive algorithm runs in a completely distributed manner. Based on a queueing model, the algorithm utilizes local information to estimate the overall network condition and switches the node state between Compression and No-Compression according to the potential benefit of compression. The enhanced algorithm further utilizes some extra information from other nodes to obtain more accurate estimation. Our extensive experimental results show that the proposed adaptive algorithm can fully exploit the benefit of compression while avoiding the potential hazard of compression. Finally, it should be pointed out the proposed adaptive algorithm is not restricted to the LZW compression algorithm, and in fact, it can be applied to any practical compression algorithms by simply replacing the compressor in ACS.

Chapter 4

Communication Synchronization in Cluster-Based Sensor Networks

Clustering is one of the most suitable structure to support large-scale wireless sensor networks. In this chapter, we consider the communication synchronization problem in cluster-based sensor networks with delay requirements. We propose a hybrid scheduling scheme that integrates a synchronous scheduling algorithm that achieves low packet delay with high energy efficiency, and an asynchronous algorithm that achieves lower packet delay with great scheduling flexibility.

The chapter is organized as follows. Section 4.1 and Section 4.2 discusses the background and related works. Section 4.3 and Section 4.4 describe the design of synchronous and asynchronous approaches, respectively. The integration of the two approaches is then elaborated in Section 4.5. Section 4.6 presents the experimental results for the proposed approaches. Finally, Section 4.7 concludes the chapter.

4.1 Introduction

The emerging Cyber-Physical System (CPS) has been gradually changing the society and the world by interacting with people's everyday life. Its research and use can be found in various applications of different societal services, including efficient energy control systems, intelligent traffic monitoring, medical care system and etc. [65]. In general CPS, information of physical world is collected and analyzed for the computing system to make appropriate decisions and controls responding to any physical situations and changes. While there are design challenges in different aspects of CPS, we focus on providing a fundamental infrastructure for information collection in CPS.

Wireless sensor network, as originally designed to perform sensing tasks, is a natural fit for information collection in CPS. Compared to conventional WSNs, WSNs adopted in CPS faces more stringent design challenges.

- WSNs should have good scalability to adapt to the increasing geographical range covered by CPS.
- WSNs should be performed in a low-latency fashion to support real-time interaction with the physical world.
- WSNs should still be energy efficient to support long and stable service for CPS. Notice that previous energy efficient solutions in WSNs may not be suitable in CPS as they may not satisfy the other two requirements simultaneously.

Of all kinds of topologies in WSNs, clustering is a good candidate to meet the above challenges, considering its wide use in WSNs to increase scalability, improve energy efficiency and provide QoS guarantees. With clustering, sensor nodes are organized into clusters and a cluster head (CH) node is selected for each cluster according to certain rules, while other nodes act as members in the clusters. In cluster-based data gathering, data collected by cluster members are first sent to CHs, which in turn deliver the data to the data sink either by direct communication or through relays on intermediate CHs. While clustering is initially introduced to achieve energy efficiency, it can also help maintain low packet latency in delay-sensitive data gathering. This is because that packets from different members can be combined as aggregated packets at CHs to reduce the transmission overhead of packet headers and control packets (e.g., ACK packets), leading to shortened transmission delay and increased energy efficiency. In addition, clustering simplifies the routing from the source node to the sink, and shorter routing paths reduce network traffic as well.

To support low-latency data gathering, however, cluster-based WSNs encounter a new communication synchronization problem due to their more complex communication patterns compared to WSNs with a flat topology. In general, the communication in a cluster-based WSN includes intra-cluster communication among sensors in the same cluster and inter-cluster communication among different CHs. Intra-cluster communication in each cluster is usually controlled by the CH with a Time-Division-Multiple-Access (TDMA) based protocol to avoid transmission collisions. For inter-cluster communication, CHs can be considered to form a smaller *relay network* where either TDMA or Carrier-Sense-Multiple-Access (CSMA) based protocols can be utilized. To avoid interference between intra- and inter-cluster communications, different channels are used for two types of communications, which implies that CHs have to switch between two channels accordingly as most of sensors can operate on only one radio channel at a time. Let *i-state* and *o-state* denote that a CH uses the channel for intra-cluster communication and inter-cluster communication, respectively. Such state switching thus incurs a synchronization problem which is critical for delay-sensitive applications: the sending CH and the receiving CH should be in o-state simultaneously and any inter-cluster packet transmission may endure an unacceptable long delay before the receiver switches to o-state. In such a case, the inter-cluster packet that consists of multiple sensing packets may become useless and be discarded, causing severe performance loss.

Previous work that targets at the energy efficiency of clustering handles this synchronization problem by simply grouping the intra- and inter-cluster communications involved in all clusters into two global and non-overlapping periods. In this approach, since the intra- and inter-cluster communications are guaranteed to be performed separately, the synchronization problem can be avoided. However, by intentionally separating two types of communications, such an approach may cause low channel utilization and hence long end-to-end delays, rendering it not suitable for delay-sensitive applications. In this paper, we first propose two communication scheduling approaches to solving the synchronization problem from different angles and support delay-sensitive data gathering applications with different requirements. We then discuss the integration of the two approaches and its suitability to use in CPS.

We first propose a TDMA based, synchronous scheduling approach to achieve low end-to-end packet delay by converting the cluster synchronization problem into a scheduling problem in generic wireless networks. Due to the NP-completeness of the scheduling problem, we propose an efficient heuristic scheduling algorithm. Compared to other cycle-based approaches in the literature, our approach owns three unique features. First, the cluster heads can individually decide their intra-cluster packet collection time, rather than a globally synchronized collection time. Second, since all packets are sent to the sink, we schedule transmissions according to the order of nodes in the routing path to minimize the queueing delay. Third, we efficiently overlap the transmissions so that the cycle length and the average end-to-end packet delay can be reduced as much as possible. Experiments show our approach can achieve 50% shorter average packet delay than the existing approach.

The cycle based approach may have some restrictions in a harsh environment due to its vulnerability to the clock drift, topology changes and irregular interferences occurred in WSNs [28]. We thus propose a CSMA based approach that solves the communication synchronization problem asynchronously. The approach is constructed on a new clustering structure with a new type of node, called *relay node*, rather than the conventional CH-member structure. The relay nodes stay in o-state and replace the CHs to receive and forward the aggregated packets. With the assistance of relay nodes, inter-cluster communications are automatically synchronized. Compared to the first approach, the asynchronous approach better utilizes the wireless channel and yields even lower packet delay. The performance of both approaches has been verified through extensive ns-2 simulations.

While these two approaches can fulfill different performance requirements, they are then integrated as a hybrid scheduling scheme that fully exploits the benefit from both approaches and allows dynamic switching among them to adapt to various network conditions. When emergency occurs or wireless signal quality if poor, the second approach is preferred to enjoy lower latency and higher interference tolerance; when the emergency ends, the first approach can be used to achieve higher energy efficiency based on the nature of TDMA. We believe the hybrid approach is suitable to support the fundamental network for information collection in CPS.

4.2 Related Work

Clustering is a popular topology control approach to achieving energy efficiency and scalability in WSNs. In this section, we briefly review the cluster formation algorithms and then discuss some existing work concerning communication protocols in cluster-based networks.

Cluster formation algorithms have been extensively studied in the literature. Their primary purpose is to consider load balance and energy efficiency to prolong network lifetime. While some algorithms consider a heterogeneous environment where CHs are more powerful than regular sensor nodes [66, 69], other algorithms consider a homogeneous environment where CHs are ordinary sensors [70]. In this chapter, we mainly focus on clustering in a homogeneous environment. Typical algorithms in this category include LEACH [67] and HEED [68] LEACH selects CHs randomly and distributes energy consumption evenly among all nodes by cluster head rotation. HEED selects CHs by considering the residual energy in the nodes and the communication cost. A comprehensive survey on different clustering algorithms can be found in [70].

There is also some work on communication protocols in cluster-based networks. While intra-cluster communication in these protocols is always TDMA-based, the inter-cluster communication adopts different approaches in different works. In [71], a round-based data collection scheme with direct sink access was proposed. It was assumed that CHs can directly access the sink, which has the capability of multiple packet reception. The intra- and inter-cluster packet delay was considered separately and the end-to-end delay was not studied. A similar round-based protocol was proposed in [72], where the routing path for inter-cluster communication consists of either cluster heads or a combination of CHs and members. In [73], a pure TDMA-based scheme was proposed to achieve optimized energy efficiency and minimum delay, in which the packet delay is directly associated with the length of TDMA frame. In addition, the MAC protocol defined in IEEE 802.15.4 can be utilized in cluster-based networks [74]. In particular, a cluster-tree topology is constructed with each CH corresponding to a coordinator, which maintains a superframe with 16 slots. The members are allowed to communicate with the CH in any slot in the superframe. In general, superframes for different coordinators do not overlap so that the interference among different clusters can be avoided.

In the above protocols, communication synchronization is handled by either setting a complete transmission schedule for every CH, or globally separating the intra- and inter-cluster communications. As will be seen in the performance evaluation section later, compared to our proposed approaches, these approaches do not perform well in delay-sensitive data gathering.

At last, many communication protocols in cluster-based networks adopt hybrid approaches that utilize both TDMA and CSMA. Such hybrid approaches are also commonly seen in general WSNs, such as S-MAC [75], T-MAC [76], Z-MAC [83] and funneling-MAC [84]. S-MAC maintains continuous duty cycles and employs CSMA in each cycle for transmissions. T-MAC follows a similar hybrid approach and improves S-MAC in terms of energy consumption by using a listening window at the beginning of each cycle. Z-MAC tries to pertain the advantage of both TDMA and CSMA such that the hybrid approach acts like CSMA with light traffic and behaves as TDMA when traffic becomes heavier. This goal is realized by assigning ownership to each time slot and giving the owner higher priority to access the channel. funneling-MAC also utilizes the hybrid approach to solve the funneling problem, i.e., nodes closer to the sink have much heavier traffic and need more communication control. Therefore, TDMA is used for these nodes to avoid frequent contentions while CSMA is used for other nodes with less contentions. These protocols, however, are designed for general WSNs and cannot be directly applied in cluster-based WSNs.

4.3 Synchronous Scheduling

In this section, we present Cycle-Based Scheduling (CBS), a TDMA based, synchronous scheduling approach. To begin with, we describe the assumptions for the system, which are also shared by the asynchronous scheduling approach.

4.3.1 Assumptions

The network considered is a WSN with n sensor nodes randomly distributed in a 2D region. We consider a typical data gathering application in WSNs where all sensor nodes send collected data to a single sink. We also make the following assumptions on the WSN.

- The clustering topology is pre-constructed by a clustering algorithm, such as the algorithms mentioned in Section 4.2, which indicates that the size of the different clusters may be different. In addition, we assume the topology of the relay network is stable during the data gathering. This is reasonable if the CHs are properly selected with adequate energy.
- Sensors can transmit on different radio channels. However, they can only transmit or receive packets on one channel in any instant. Different radio channels do not interfere with each other.
- Sensors have the same sensing rate λ and sensing packets are of the same length. The packet generation process is assumed to be Poisson.

Under the above assumptions, which are applicable in many real-world networks, next we describe the details of CBS.

4.3.2 Basic Communication Cycle

CBS schedules communications in consecutive cycles and each node is assigned some fixed conflict-free intervals to transmit and receive packets in each cycle. Nodes only wake up in the assigned intervals and sleep otherwise to reduce energy consumption. Each node is assigned a single interval for transmission so that the synchronization overhead between the transmission pair is minimized. The goal of the scheduling is to minimize the average end-to-end packet delay. We consider this problem by separating the intra- and the inter-cluster communications.

Intra-cluster communication

Intra-cluster communication includes all transmissions from cluster members to the CH. Since interferences from other clusters can be avoided by assigning different radio channels to adjacent clusters, the communications within a cluster are independent and hence it is reasonable to only consider a single cluster.

We limit all communications for a cycle in a consecutive period so that the CH needs not to frequently switch between intra- and inter-cluster communications. As will be seen later, the duration of this period is relatively short compared to the cycle length, we simply consider a general TDMA scheme for the intra-cluster period. The whole period is divided into multiple identical time slots whose length τ is set equal to the time required for a packet transmission. Packets are sent in these time slots directly from cluster members to the CH. Each node is assigned the same k time slots given their same packet generation rate. For simplicity, we assume the CH is also assigned k time slots for necessary control packets. Assume the cluster has m nodes, the duration of the intra-cluster period is thus $m \cdot k \cdot \tau$.

For such scheduling, we are concerned with the determination of k and the packet collection delay, which is defined as the elapsed time between the packet is generated and the end of the intra-cluster period in which the packet is collected.

Lemma 1. *The lower bound of k is $\lceil \lambda T \rceil$, where T is the cycle length.*

Proof. The expected number of packets generated by each node in a cycle is λT. Since a cluster member can transmit one packet in a time slot, in order to collect all packets in one intra-cluster period, it must satisfy $k \geq \lambda T$. Since k can only be an integer, the lower bound of k is $\lceil \lambda T \rceil$. □

Lemma 2. *If k is large enough for collecting all packets in a cycle, the expected collection delay is $\frac{T+m\cdot k\cdot\tau}{2}$.*

Proof. Consider a packet generated by node i, whose time slots assigned end at s_i. Since k is large enough for collecting all packets in a cycle, this packet must be generated between the end of slot s_i of two consecutive intra-cluster periods and this interval is T. Since the packet generation process is Poisson, the time a particular packet is generated within a fixed interval is uniform [81], thus the expected generation time in this interval is $\frac{T}{2}$ and the expected collection time is $\frac{T}{2} + (m \cdot k - s_i)\tau$. Therefore, the expected collection time for all packets will be

$$\begin{aligned} E(D_c) &= \frac{T}{2} + m \cdot k \cdot \tau - E(s_i) \\ &= \frac{T + m \cdot k \cdot \tau}{2} \end{aligned} \tag{4.1}$$

□

Lemma 2 indicates that the intra-cluster period can be placed at any position in the cycle without affecting the collection delay, which is only dependent on the cycle length and the period duration. It also suggests that k can be selected at its lower bound $\lceil \lambda T \rceil$ to minimize the collection delay, which monotonically increases with k.

Inter-cluster Communication

Inter-cluster communication includes transmissions in the relay network, which consists of CHs and the sink. For data gathering, as the relay network is stable, the CHs are organized into a fixed routing tree rooted at the sink at the same time when the clusters are formed. The construction of the routing tree is independent of our scheduling approach and thus is not discussed in this chapter. Within a cycle, each CH is assigned an interval to send all packets, including packets collected by itself and packets received from other CHs, to its parent. The practical length of this interval should be slightly longer than the transmission time of all packets to accommodate the necessary control packets such as ACK and potential synchronization errors. However, since we are focusing on the cycle scheduling, we set the length equal to the transmission time of all packets for simplicity.

The relay network can be viewed as a general wireless network except that the CH is not available during intra-cluster period. We introduce an *intra node* for each CH to represent the intra-cluster packet collection. Intra node i generates $m_i \cdot k$ packets in each cycle with zero queueing delay before the packets are sent out. It transmits packets to the CH within an interval whose duration equals $m_i \cdot k \cdot \tau$. The transmission does not affect other nodes except for the associated CH. Thus the considered problem becomes to schedule intervals for all nodes in the transformed network.

4.3.3 Interval Schedule

To obtain an efficient interval schedule, we first present an analytical model for the problem and then show an illustration example. Guided by the example, we will propose our scheduling approach.

Mathematical Model

The network is represented by a graph $G = (V, E)$. V is the set of nodes, including the sink, the CHs and the corresponding intra nodes. Denote p_i as the parent of node i in the routing tree and (i, j) as a transmission link between node i and node j, then $E = \{(i, p_i) | i \in V\}$. Since every node has a fixed parent, it is easy to see $|V| = |E| + 1$.

To model the interference of transmissions, we construct a conflict graph $G' = (V', E')$. V' represents all the transmission links in E. For simplicity, we use i to represent link (i, p_i) such that $V' = V \backslash \{v_s\}$, where v_s represents the sink. E' is constructed such that if $(i, j) \in E'$, nodes i and j cannot transmit at the same time due to that the distance between any two of nodes i, j, p_i and p_j is within the transmission radius. Such construction is valid if we assume that the receiver may send ACK packets and transmissions will not only be affected by the sender, but also by the receiver. For an intra node i, there is only one conflict edge (i, p_i) corresponding to the fact that the CH cannot send packets during the transmission of its corresponding intra node.

The interval scheduling problem is to find a feasible time interval (s_i, f_i) for each node i in V', where s_i and f_i are the starting and finishing time instant with $0 \leq s_i \leq f_i$. The cycle length is then set as $t = \max_{i \in V'} f_i$. Here we normalize the cycle into time slots with length $k \cdot \tau$ so that the actual cycle length $T = k \cdot \tau \cdot t$. Since the interval equals the transmission time of all packets, we have $f_i = s_i + n_i$, where n_i is the maximum number of packets node i sends in a cycle and can be obtained by

$$n_i = \begin{cases} m_i & \text{if } i \text{ is an intra node} \\ \sum_{j \in C_i} n_j & \text{if otherwise} \end{cases}$$

Here, C_i denotes the child set of node i. For an interval of node i to be feasible, its transmission link should not conflict with any other transmission links, thus $\forall (i, j) \in E'$, $s_i \geq f_j$ or $s_j \geq f_i$.

The end-to-end packet delay can be broken down into transmission delay and queueing delay. The transmission delay from an intra node to the corresponding CH is simply the collection delay in the intra-cluster period while the transmission delay from CH i to its parent p_i is n_i. The queueing delay, defined as the waiting time of a packet at a node before it is sent out , will be $(s_{p_i} - f_i + t) \mod t$ for a parent p_i. Thus the average packet delay is

$$D = k \cdot \tau \cdot \frac{\sum_{i \in V'} n_i (d_i + (s_{p_i} - f_i + t) \mod t)}{n_{v_s}} \tag{4.2}$$

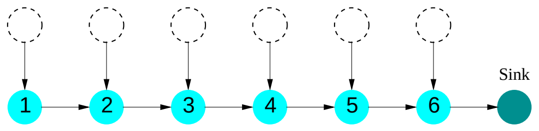
where

$$d_i = \begin{cases} \frac{t+m}{2} & \text{if } i \text{ is an intra node} \\ n_i & \text{if otherwise} \end{cases}$$

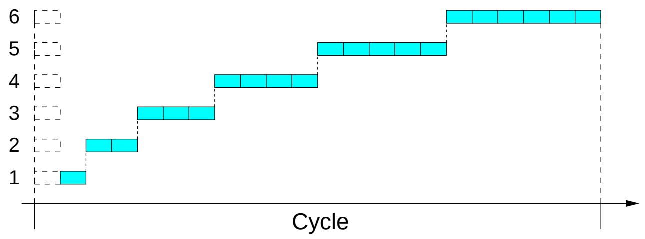
The optimal scheduling problem was proved to be NP-complete by reducing the K-Colorability problem to the scheduling problem [82]. Therefore, we will design a heuristic algorithm. Before that, we examine an example to reveal some interesting property in the scheduling problem.

Example of Chain Topology

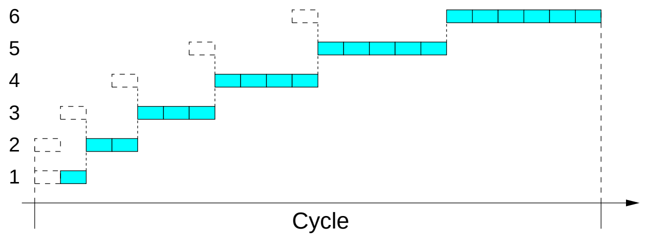
This example considers a network with chain topology as shown in Fig. 4.1(a). Each CH has a corresponding intra node, which generates one packet in a cycle. Thus, node i will send i packets in a cycle.



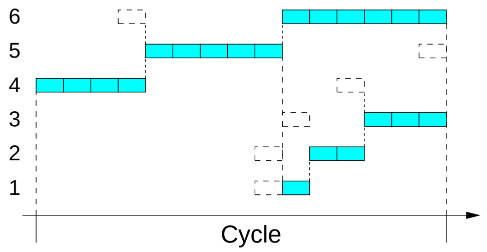
(a) Chain network consists of 6 CHs.



(b) Intuitive scheduling



(c) Improved scheduling



(d) Optimal scheduling

Figure 4.1: An example network with chain topology.

Fig. 4.1(b) shows an intuitive scheduling, where node i is sequentially assigned an interval of i and all intra nodes are assigned an interval of 1 at the beginning of the cycle. This is actually a not-so-bad scheduling as it eliminates the queueing delay in the relaying: a CH will immediately send out all the packets upon its receiving from its child. The average packet delay $D = 32.5$.

The intuitive scheduling does not consider the fact that packets have zero queueing delay on the intra node. An improved scheduling in Fig. 4.1(c) schedules the intervals for intra nodes as close as possible to the corresponding CHs. The average packet delay $D = 29.2$.

In the improved scheduling, the collection delay on intra node, which is 11.5 according to Eq. (4.1), takes a large part in the total delay. Since the delay mainly depends on the cycle length, we can further reduce the average packet delay by reducing the cycle length. Fig. 4.1(d) shows the optimal scheduling that minimizes the cycle length. In the scheduling, the cycle length is exactly the sum of the interval lengths of nodes 4, 5 and 6. Since these nodes are conflicting with each other, their intervals cannot overlap and thus the cycle length reaches its minimum. On the other hand, the intervals of nodes 1, 2 and 3 are placed at the end of the cycle so that no extra queueing delay is introduced. The average packet delay $D = 25.7$.

Following this example, we obtain three guidelines to design the scheduling algorithm.

- Interval assignment should follow the order of the nodes in the routing path.
- Intervals for intra nodes should be as close as possible to the intervals for the corresponding CH.
- The cycle length can be reduced by overlapping as much intervals as possible. However, the reduction is bounded by the intervals that cannot be overlapped due to the conflicts. In fact, although each interval will have a number of conflicted intervals, it is the longer intervals that decide the lower bound of the cycle length.

Algorithm

The algorithm is summarized in Tables 4.1 and 4.2. Table 4.1 describes a basic scheduling algorithm that strictly schedules the nodes according to their orders in the routing path: a node is always scheduled before its parent. From the second guideline, we see that nodes should be scheduled as late as possible. Thus, the algorithm actually schedules nodes in the reverse order so that nodes can be scheduled closer to the end of the cycle. For illustration, we define function $I(\cdot)$ on nodes such that

$$I(i) = \begin{cases} 1 & \text{if } i \text{ is scheduled} \\ 0 & \text{if } i \text{ is not scheduled} \end{cases}$$

During the scheduling, a set V_t is maintained to include nodes whose parents are already scheduled. The algorithm tries to find the earliest possible scheduling interval for all nodes in V_t and the node with the earliest starting time is scheduled. Then the algorithm schedules the next node with no earlier starting time until all nodes are scheduled or no intervals can be scheduled within the required range. Then the reverse schedule is obtained.

Table 4.1: Basic Scheduling Algorithm

```
**Input:** node set to be scheduled $V_s$, schedule range $(S, F)$
**Output:** updated schedule
$V_t = \{i | i \in V_s, I(i) = 0, I(p_i) = 1\}$
**while** $V_t! = \emptyset$
    order $V_t$ by distance to sink
    find earliest schedulable interval $(s_i, f_i), s_i \geq S$
        for each node in $V_t$.
    $j = \arg\min_{i \in V_t} s_i$
    **if** $f_j > F$ **return**
    schedule node $j$ with $(s_j, f_j)$
    $S = s_j$
    update $V_t$
**end while**
```

Table 4.2 utilizes this basic algorithm to perform the actual scheduling. The idea is to first determine a tentative cycle length and then try to schedule all the intervals within this cycle. Given the third guideline, we start to schedule the intervals from the nodes that are closer to the sink. For assistance, we construct two node sets V_n and V_c. Let

$$V_n = \{i | I(i) = 0, I(p_i) = 1, i \in V'\}.$$

Initially we assume $I(v_s) = 1$ so that V_n includes all nodes that directly send packets to the sink. Clearly, these intervals cannot be overlapped. In addition, their conflicting intervals cannot be overlapped with these intervals either. For that, we construct

$$V_c = V_n \cup \{j | (i, j) \in E', I(j) = 0, i \in V_n\}.$$

We then schedule V_c with the basic scheduling algorithm with no range requirement and obtain the tentative cycle length. For other nodes that are not scheduled, since they are not in the current V_c, it is guaranteed that their scheduled intervals can be overlapped with intervals for nodes in V_n. Thus we can schedule the rest of nodes from the beginning of the cycle. Thus we update V_n and V_c according to current schedule and repeat the basic scheduling algorithm. Since nodes that are closer to the sink have longer intervals, in most cases the updated V_c can be scheduled at the beginning part of the cycle, leaving the rest part of the cycle available for further scheduling. We then schedule the rest of nodes to fill in the available part of the cycle to avoid queueing delays. This process is repeated until all nodes are scheduled. The finiteness of this process is guaranteed by the construction of V_n, which guarantees that all children of already scheduled nodes will be scheduled in the next iteration. In fact, our experiments show that two iterations will suffice in most cases as the tentative cycle length is large enough for the rest of nodes to schedule sequentially. Notice that since the basic scheduling algorithm schedules nodes reversely, the actual schedule should be in the exactly reverse order of the obtained schedule.

Analysis

With this cycle-based scheduling, we are also interested in determining the maximum packet generation rate. Recall $k \geq \lambda T$. We have

$$\begin{aligned} k &\geq \lambda T = k \cdot \lambda \tau t, \\ \lambda &\leq \frac{1}{\tau t}. \end{aligned}$$

Therefore, the maximum packet generation rate is $\frac{1}{\tau t}$. On the other hand, when the generation rate does not exceed the maximum rate, it is always satisfied that $k \geq \lambda T$. Thus, we can always set $k = 1$ to minimize the packet delay.

4.4 Asynchronous Scheduling Approach

In this section, we present the second scheduling approach, which is called New Cluster Scheduling (NCS), adopts an asynchronous approach that essentially

Table 4.2: Actual Scheduling Algorithm

```
**Input:** graph $G = (V, E)$ and conflict graph $G' = (V', E')$.
**Output:** interval schedules for nodes in $V'$
$t = 0$
construct $V_n$, $V_c$
while $V_c! = \emptyset$
    schedule $V_c$ with range $[0, \infty)$
    $t_c = \max_{i \in V_c} f_i$
    if $t_c < t$
        $V_u = \{i | i \in V', I(i) = 0\}$
        schedule $V_u$ with range $[t_c, t]$
    end if
    $t = \max(t, t_c)$
    update $V_n$, $V_c$
end while
reverse the whole schedule
```

avoids the synchronization problem by introducing a new clustering structure. Next we first introduce the new clustering structure and then describe the approach in detail.

4.4.1 New Clustering Structure

Instead of designing another algorithm to globally schedule all the inter-cluster communications for synchronization, NCS attempts to simplify the synchronization by changing the communication pattern. To achieve this goal, NCS introduces a new clustering structure, which includes a new type of node: relay node.

The new clustering structure is illustrated in Fig. 4.2, in which a cluster contains a CH node, a relay node and multiple cluster members. The relay nodes always stay in o-state and only participate in inter-cluster communications. During data gathering, while cluster members still send sensing packets to the corresponding CH, the CH no longer sends the aggregated packet to the next-hop CH but sends to the relay node of its own cluster instead. Upon receiving the packets, the relay node further combines them with its own sensing packets and forwards the packets to the next-hop relay node until the packets reach the sink. With such communication pattern, the communication synchronization is greatly simplified. CHs can continue intra-cluster data collection immediately after sending out the aggregated packet, reducing the data collection delay. In the meanwhile, inter-cluster communication can be performed without any restrictions, incurring no waiting delays for synchronization. The wireless channel thus can be better utilized and lower packet delay can be achieved.

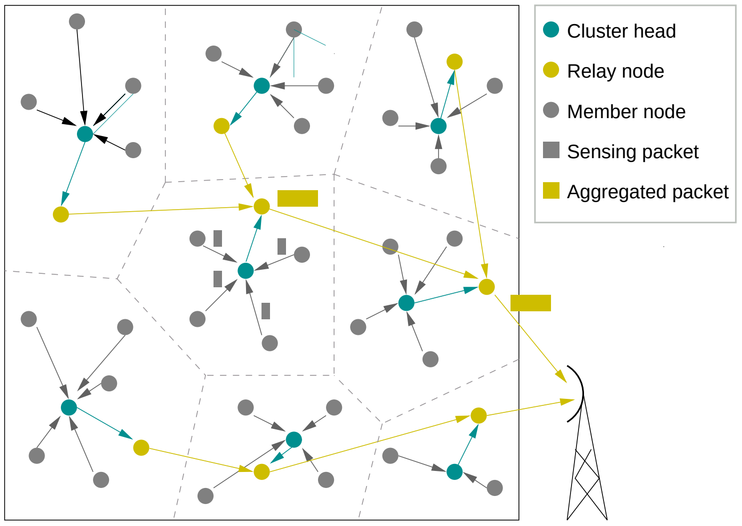


Figure 4.2: The new clustering structure includes CHs, relay nodes and members. The packet transmissions for the center cluster are shown.

On the other hand, the new clustering structure does not substantially increase the complexity of the cluster formation process. A network with the new clustering structure can be simply converted from a network with the conventional clustering structure by selecting the member with the highest residual energy as the relay node in each cluster. The routing algorithms for creating routes among different CHs in conventional cluster-based networks can also be utilized to create routes among CHs and relay nodes.

4.4.2 Approach Details

NCS adopts the same TDMA protocol as CBS for intra-cluster communications and CSMA protocol for inter-cluster communications. While member nodes and relay nodes are fixed in i-state and o-state, respectively, CHs still need to switch between two states, which is the major task of NCS. Since there is no synchronization required among different CHs, the state switching, or the duration at each state, can be determined independently for each CH.

We first determine the inter-cluster duration a CH stays in o-state. When a CH switches to o-state, it cannot transmit a packet immediately. Consider the case that CH 1 is sending a packet to CH 2 at o-state while CH 3, a neighbor of CH 2, switches to o-state. If CH 3 is not in the transmission range of CH 1, it will not detect the ongoing transmission, which would be interrupted by any transmission initiated at the CH before the end of the current transmission. In this case, the protection from RTS/CTS handshake fails as they were not received by the CH who was in i-state then. We call the period in which such collisions may occur the *blind* period and its duration equals the transmission time of a packet of a maximum allowable packet length. After this blind period, the CH then sends the aggregated packet to the next-hop relay node on the routing path. Once the transmissions are completed, the CH can immediately switch back to i-state to continue data collection.

Next we consider the intra-cluster duration of a CH in i-state. Since the CH does not participate in inter-cluster communications for other CHs, the duration in i-state only affects its own collection delay. Intuitively, to minimize the collection delay, the CH can switch to o-state immediately after the end of the time frame in which a packet is collected. However, such an approach yields relatively small aggregated packets, which underutilizes the wireless channel due to the overhead of packet headers and control packets, lowering the maximum achievable throughput. Alternatively, we use a fixed collection duration, denoted as T_c. A larger T_c indicates less frequent data collection, yielding a smaller number of larger aggregated packets. Consequently, the channel is better utilized and higher throughput can be achieved. On the other hand, a larger T_c also leads to longer collection delay and hence the end-to-end packet delay. Therefore, adjusting T_c can obtain different tradeoffs between the packet delay and the maximum achievable throughput.

T_c determines the number of time slots in an intra-cluster period. Following a similar analysis to that in Section 4.3.2, we see that the necessary number of time slots for a member in an intra-cluster period is $k = \lceil \lambda(T_c + T_o) \rceil$, where T_o represents the duration of the last inter-cluster period. When $T_c > m \cdot k \cdot \tau$, a portion of the intra-cluster period is actually wasted. For energy efficiency, we organize the intra-cluster period into time frames with each consisting m time slot, allowing each node to send a packet in a time frame. Then the CH can remain active only in the last k frames and sleep in other times. The entire process for a CH is described in Fig. 4.3.

4.4.3 Delay Guarantee

Thanks to the relay nodes, NCS avoids the synchronization delays during the inter-cluster communications, allowing the relay network to operate similarly

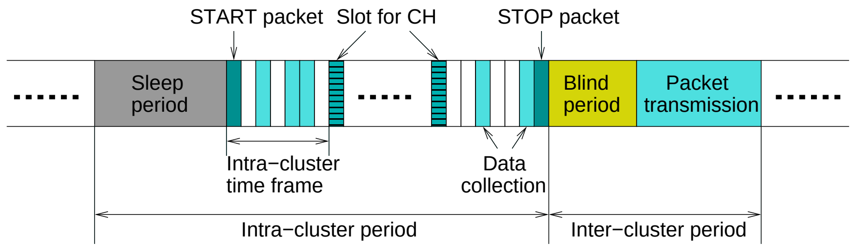


Figure 4.3: Timing of data gathering at a CH in NCS.

to a general WSN. Therefore, although it does not directly provide delay guarantees, it greatly facilitates the utilization of real-time routing protocols at the upper layer, such as SPEED [29] and MMSPEED [30], which relies heavily on the one-hop packet delays. These delays could be very long and irregular in a cluster-based network where communications incur synchronization delays, degrading the performance of the real-time routing protocols. On the contrast, with NCS eliminating the synchronization delays, real-time routing protocols can be easily implemented to provide optimal performance on delay guarantees.

4.5 Integrating CBS and NCS

CBS and NCS are designed to satisfy different network requirements, with each having its own tradeoff. CBS is more energy efficient due to the nature of TDMA: nodes can sleep in any idle slots to preserve energy. On the other hand, NCS yields lower end-to-end packet delay as will be shown in the evaluation in Section 4.6. Moreover, the CSMA based protocol makes NCS more tolerant to interference and collisions. In a general CPS application, we envision that the targeted emergencies do not occur often. As a result, CBS will be sufficient to monitor the environment in most time, and the network can switch to NCS to track the emergency for prompt reaction. Thus, A hybrid scheduling scheme that integrates CBS and NCS can meet the CPS requirements of both achieving low latency when necessary and preserving a long life time.

The major challenge in the hybrid scheme is the switching between the adoptions of CBS and NCS. Intuitively, scheduling switching can be performed concurrently with the next round clustering which reconstructs clusters. However, such reconstruction is both energy and time consuming, causing interrupts on the current monitoring task as well. Therefore, the switching should be smoothly performed on the current clusters, which we discuss in this section.

4.5.1 Switching from CBS to NCS

The switching from CBS to NCS occurs when the emergency or severe channel interference is detected by the sink through the analysis of the received data. This process can be divided into two major tasks: to adapt the current cluster structure to NCS while maintaining the connectivity, and to notify all clusters to perform the switching smoothly.

Cluster Structure Adaption

Switching to NCS requires a node in each cluster to be selected as the relay node. Such selection should be very carefully done since in NCS, the relay nodes are responsible for the connectivity of the relay network. In practice, we select the current CH as the relay node for each cluster so that the topology of the relay network keeps unchanged and the network naturally remains connected during the switching. Such selection also eliminates the potential updates of the inter-cluster routing information for all clusters, minimizing the inter-cluster communication overhead. In addition, maintaining a connected relay network during the switching facilitates the possible cluster reorganization: in each newly constructed cluster, the CH just need to find a relay node to keep the cluster connected. We do not further articulate the cluster reorganization process as it is beyond the scope of the chapter.

With the CH becoming the relay node, we select the node with the highest residual energy from the remaining nodes as the new CH. A new intra-cluster scheduling can then be easily performed according to the specification of NCS.

Switching Notification

Switching is initiated by the sink, which spreads the switching notification in the reverse order of data collection: the sink and the CHs are responsible for notifying their direct children. The notification can be simply piggybacked in the ACK packet when a data packet is received from an un-notified child. The notified child can then switch to NCS accordingly and notify its own children in the next cycle until the switch is completed on the whole network.

4.5.2 Switching from NCS to CBS

When the sink decides that the network can return to regular monitoring, a switching from NCS to CBS is necessary to increase energy efficiency. Similarly, we also consider the two major tasks in the switching: cluster structure adaption and switching notification.

The more challenging task is the cluster structure adaption, which is essentially the CH selection problem for CBS. Letting the relay node or the current CH in NCS become the next CH in CBS can have certain advantages. Selecting the relay node can easily maintain the network connectivity while selecting the current CH can maintain the current TDMA scheduling with a slight modification of adding a slot for the relay node. However, both may not be the optimal decision from the perspective of energy efficiency. Since the relay node and the CH in NCS both are major energy consumers, their residual energy may not be sufficient for the subsequent monitoring. In this case, selecting them as the CH can deplete their batteries quickly, causing connection failure and subsequent clustering reconstruction. Therefore, we propose a CH selection algorithm to determine the adaption.

CH Selection

Through the CH selection, we attempt to maximize the remaining network lifetime while maintaining network connectivity. The remaining lifetime can be defined as the interval between the current instant and the time when the first CH exhausts the energy and a cluster reconstruction is needed. Achieving this goal requires global optimization because locally selecting the node with the highest residual energy as a CH for each cluster cannot guarantee the connectivity. A possible solution may be to gather the energy of all nodes at the sink, calculate the CH selection for all clusters and disseminate the selection to all clusters. However, the information gathering and dissemination cost both time and energy; the centralized algorithm itself is inflexible to adapt to the network dynamics. Alternatively, we propose a heuristic approach to determining CHs distributively. This distributed algorithm may not obtain global optimization due to lack of global information, however, it can achieve a slightly modified goal: ensure network lifetime longer than some threshold. This threshold can be set to be the expected duration before the next switching so that the network can be alive as long as this goal can be achieved in every switching from NCS to CBS.

Guaranteeing the global connectivity by local CH selection is not an easy task. Recalling that a routing tree must be determined before CBS can be used, we perform the CH selection based on the routing tree. For a typical cluster that needs to select a CH h, a parent list is maintained as $P = \{p_1, \dots, p_k\}$, in which any node could be potentially the parent of h in the routing tree. The construction of this list will be described later in Section 4.5.2. A cluster also records a child list $\mathcal{C} = \{C_1, \dots, C_l\}$, where C_i is a cluster whose CH is potentially a child of h in the routing tree. This child list is created before the switching. Each relay node randomly chooses a next-hop relay node on the routing path as a candidate parent, which in turn adds this relay node in the child list. In addition, the cluster should know all the neighboring nodes that can communicate with any node in the cluster, which can be obtained by exchanging information at the cluster creation stage. Finally, the cluster is aware of the residual energy of all cluster members, which can be easily obtained by the CH.

With the above information, we can now describe the algorithm, which is run on the relay node. The notations used in the algorithm are listed in Table 4.3 and the pseudo code is listed in Table 4.4.

Table 4.3: Notations used in the CH selection algorithm

Notation	Definition
N	Node set of the current cluster
P	Parent list
$\mathcal{C}$	Child cluster list
t_l	Lifetime threshold
ϵ	Energy consumption in unit time
e_i	Residual energy in node i
B_i	Neighbor set of node i
w_i	Weight of node i

The algorithm starts with the selection of the parent CH. The relay node randomly selects a candidate parent from the parent list P (line 2). Given the selected parent, the candidate CH can only be a node that can communicate with the parent. Meanwhile, to ensure the remaining network lifetime to be longer than the threshold t_l, the residual energy in the candidate should be more than $\epsilon \cdot t_l$, where ϵ is the energy consumption in a unit time. Thus we can obtain the candidate CH set by $S = \{i | i \in N \cap B_p, e_i > \epsilon \cdot t_l\}$ (line 3). For each node in this set, we calculate the number of nodes this node can reach in each cluster in the child list (lines 5-7) and choose the minimum number as the weight of this node (line 8). Then the node with the maximum weight is tentatively selected as the CH (line 11). The idea behind such selection is that the more nodes this CH can reach in the child clusters, the higher the probability that the child clusters can select proper CHs to ensure the connectivity. There is also a possibility that the maximum weight is zero, indicating that there is at least one child cluster in which none of the nodes can connect to a node in the candidate set. In this case, we remove the selected parent from the parent list (line 15) and repeat the above procedure until a

Table 4.4: CH selection algorithm

```
1:  **While** $P$ is not empty **do**
2:    $p = \text{random\_select}(P)$
3:    construct set $S = \{i | i \in N \cap B_p, e_i > \epsilon \cdot t_l\}$
4:    **for** each node $i$ in $S$ **do**
5:      **for** each cluster $C_j$ in $\mathcal{C}$ **do**
6:        $w_j^i = |C_j \cap B_i|$
7:      **end for**
8:      $w_i = \min w_j^i$
9:    **end for**
10:   $w = \max w_i$
11:   $h = \arg\max_i w_i$
12:   **if** $w$ is not zero
13:     return $h$ as the CH
14:   **end if**
15:   remove $p$ from $P$
16: **end while**
```

feasible CH is selected. Otherwise, we conclude that connectivity cannot be maintained and a cluster reconstruction is necessary.

Switching Procedure

The entire switching is completed in three steps. The first step starts at the sink, which notifies all the neighboring relay nodes about the switching. For illustration purpose, we call relay nodes that take i hops to reach the sink *hop-i nodes*. These relay nodes perform the CH selection algorithm and send the selection along with the switching notification to all hop-2 relay nodes they can communicate with. The hop-2 relay nodes will then insert all the received selection in their parent list and perform their own CH selection. This iteration will continue until all relay nodes receive the notification and complete the CH selection. Notice that if a cluster selects a CH, it also determines the parent CH in the routing tree in CBS. Then after this step, the routing tree in CBS is actually determined.

The second step starts at the leave clusters with each relay node notifying the parent cluster the selected CH. Eventually the sink obtains the new clustering structure and perform the scheduling in CBS. Till now, the whole network is still operated in NCS and the third step performs the actual switching. Starting at the sink, each relay node notifies its children (if any) their scheduling and lets the new CH manage the cluster under CBS.

4.6 Experimental Evaluations

In this section we evaluate the performance of CBS and NCS and their integration through ns-2 simulations. For comparison purpose, we also evaluate two existing scheduling approaches used in cluster-based WSNs, for which we first give a brief description.

4.6.1 Compared Scheduling Approaches

The first approach we compare is a modified version of the scheduling approach used in IEEE 802.15.4, which we simply call 802.15.4 in this section. To adapt 802.15.4 in a cluster-based network, it is required to construct a cluster tree from the routing tree by adding members as the children of the corresponding CHs. Each CH or coordinator then maintains a non-conflicting superframe for its children in the cluster tree. For fair comparison, we assume the same cluster tree as in CBS are used and the length of the superframe for node i equals n_i, which is the maximum number of packets received in a cycle in CBS. The superframes are scheduled using the basic scheduling algorithm in CBS without considering the node order in the routing tree. In a superframe, we assume that each slot is collision-free so that a packet transmission in a slot never fails. When multiple children contend in a time slot, we randomly select a child to transmit while others wait for the next slot.

The second approach is a simple synchronous approach that defines a global frame for all clusters. The global frame includes intra- and inter-cluster periods. In the intra-cluster period, CHs collect data from members using the same protocol as in NCS. At the end of the intra-cluster period, all clusters enter the inter-cluster period simultaneously. The duration of intra-cluster period is also calculated in the same way as in NCS, while the duration of inter-cluster period T_o is considered as a parameter in the experiments. Although the scheduling approach is quite simple, similar ideas of this global frame have already been adopted in practice [71, 72] and we call this approach GF in the following evaluation.

4.6.2 Experiment Setup

We first describe the cluster formation algorithm adopted before elaborating other network configurations. According to the system model in Section 4.3.1, there are no restrictions on the cluster formation algorithm. For simplicity, in our experiments we form clusters based on their geographical positions, which are assumed already known to the nodes. Specifically, we assume the whole region is a unit square, which is divided into square cells with side length l and the sensors in the same cell form a cluster. The number of clusters is then $\lceil \frac{1}{l} \rceil^2$. The CH and relay node are randomly selected in each cluster. The transmission range is set to be $\sqrt{5}l$, which is the maximum distance between two nodes in neighbor cells. Such a range allows nodes within a cell or any two neighbor cells to communicate with each other and hence guarantees the connectivity in all clusters and the relay network.

To evaluate the network performance, we consider two networks with 300 nodes and 1200 nodes randomly scattered in the unit square. The sink is positioned at a corner of the square to create relatively long routing paths. The side length l is selected to be $\frac{1}{5}$ and $\frac{1}{10}$, resulting in 25 and 100 clusters, respectively. The cluster size ranges from 5 to 18. Some approach dependent parameters are listed in Table 4.5. Sensing packets have a uniform length of $30B$ and the transmission bandwidth is set to $1Mbps$. For 802.15.4 and CBS, we assume 0 header length to focus on comparison of cycle scheduling. For NCS and GF, we adopted the default header length of MAC 802.11 in ns2, which are 44B, 38B, 52B and 44B for RTS, CTS, DATA and ACK. The performance metrics evaluated are packet delay and network throughput. Packet delay is defined as the average end-to-end delay for all packets received at the sink while the network throughput can be interpreted as the maximum packet generation rate with which the network can operate steadily. The evaluation time is set to 100 seconds to obtain the network performance at the stable state. Each experiment is repeated 10 times to obtain the average value.

Table 4.5: Parameters of approaches.

Approach	Parameter	300-node network	1200-node network
NCS	T_c	$0.2s$	$2s$
GF	T_o	$2s$	$16s$

The inter-cluster communication in NCS and GF utilizes the common IEEE 802.11 MAC protocol. Practical WSNs may adopt some simplified versions of 802.11, however, the variation among these versions only affects the inter-cluster communications but does not substantially affect the overall performance evaluation. We thus adopt the default controlling packet length of MAC 802.11 in ns2, which is 44B, 38B, 52B and 44B for RTS, CTS, DATA header and ACK, respectively. For TDMA based 802.15.4 and CBS, we assume 0 header length to focus on comparison of cycle scheduling. To construct the routing tree, the CH or the relay node randomly selects a node in the neighbor cells that are closer to the sink as its next-hop. The aggregated packets at the source are not further aggregated at the intermediate nodes in the routing tree.

4.6.3 Performance Results

Fig. 4.4 shows the average packet delay among four approaches with allowable packet generation rates. The standard deviations of these delays are less than 0.02 and 0.2 for 300-node and 1000-node network respectively, indicating stable performance of the examined approaches. We first examine the result in the network with 300 nodes. We can observe that NCS yields the shortest packet delay when the network is not saturated under lower packet generation rates. Due to the introduction of relay nodes, packets are transmitted quickly at each hop without undertaking any extra delay caused by the state switching. On the opposite, GF, which also uses 802.11 for inter-cluster communications, yields the longest packet delay. In GF, the synchronization of the inter-cluster periods for different CHs incurs many concurrent packet transmissions with high contentions, eventually resulting in long delay. Two TDMA based approaches have shorter delay than GF since they completely avoid the transmission contention. However, since the transmissions in these two approaches are strictly scheduled, packets inevitably incur some queueing delay before they can be relayed by the intermediate nodes in the routing path. Thus both have longer delay than NCS. In particular, delay in CBS is about 30% shorter than that in 802.15.4, due to the efficient design of the cycle scheduling. While the cycle length of two approaches does not have much difference, packets in CBS endure less queueing delay with the ordered interval scheduling.

The performance comparison is similar in the network with 1200 nodes, where GF exhibits poor performance with allowable generation rate under 0.05, which was not shown in the figure. While NCS still shows the shortest delay, we observe that CBS obtains a higher performance gain compared to 802.15.4, whose delay is nearly twice of CBS. This contrast indicates that the scheduling in CBS enjoys more benefits in larger-scale networks.

Fig. 4.5 shows the network throughput, or the maximum packet generation rate for four approaches in both networks. GF has the worst throughput, besides its longest delay as seen in Fig. 4.4. This is because that GF requires a long inter-cluster period to accommodate long delay and therefore causes low allowable packet generation rate. NCS also has lower throughput com-

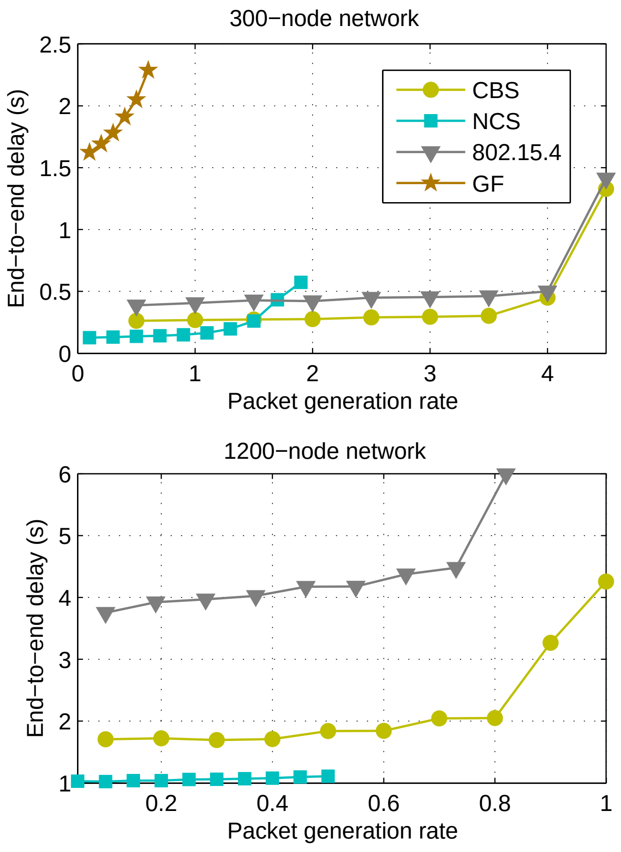


Figure 4.4: End-to-end packet delay of four scheduling approaches under different packet generation rates.

pared to two TDMA-based approaches. This is mainly due to the intrinsic of CSMA-based 802.11 protocol, which spends much longer time than the actual transmission time in transmitting packets. For two TDMA-based approaches, CBS slightly outperforms 802.15.4. The similar performance is due to the fact that the cycle length, a dominating factor for the throughput, is similar in both approaches.

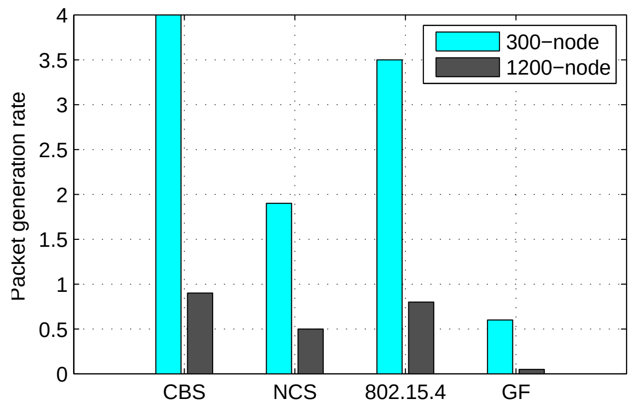


Figure 4.5: Network throughput with four scheduling approaches.

Recall in NCS, the data collection duration affects the number and length of aggregated packets and eventually the maximum achievable throughput. In the next experiment, we further reveal the relationship between the maximum generation rate and the data collection duration in Fig. 4.6. Clearly, the maximum generation rate increases faster when the duration is relatively short. Specifically, it increases about 2 times when the duration is increased from $0.1s$ to $0.5s$, and only 14% when the duration continues to increase to $1s$. Since increasing the data collection duration directly increases the collection delay, such observation indicates that the data collection duration should be chosen properly to achieve the best tradeoff between the maximum generation rate and the packet delay.

4.6.4 Integration Performance

We evaluate the integration performance in the network with 300 nodes. In the integration, we are concerned with the achievable network lifetime, which is defined as the duration between the network creation and the time instant when the relay network is no longer connected. This terminating situation corresponds to the following three cases: 1) the first node depletes its energy

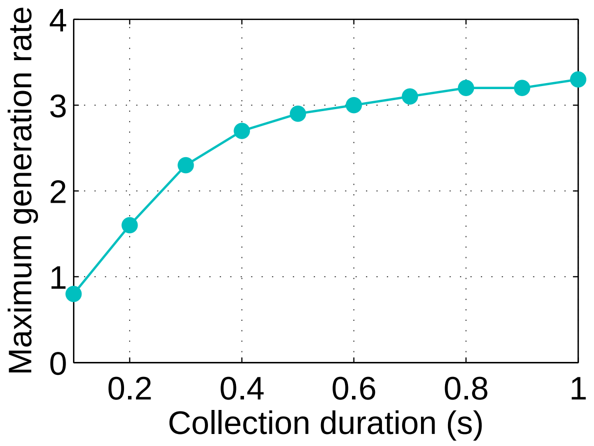


Figure 4.6: Maximum generation rate achieved with different collection durations in NCS.

during data gathering; 2) a cluster cannot complete the CH selection algorithm in Section 4.5 during the switching from NCS to CBS; 3) A relay node or a CH loses connection with its parent during the switching.

To evaluate the network lifetime, we simulate the switching by setting the consecutive time durations in CBS and NCS to be randomly distributed between 30 $\sim$ 90 minutes and 30 $\sim$ 90 seconds, respectively. The packet generation rate is fixed at 1 packet/s. To model the energy consumption, we assume the power ratio for sending, receiving and idle listening is set to 1.67:1:0.88 as adopted in [77]. In CBS, we assume each data packet is associated with an ACK packet. The length of all control packets is of default values in ns-2. The initial energy of every node is normalized to energy consumed in 24-hour continuous packet receiving. This value is reasonable if we assume the receiving power of a sensor is of order of $10mW$ and the battery energy is of order of $1000J$. Similarly, the threshold energy is set to the energy consumed in 90-minute packet receiving.

With the above parameters, network lifetime is heavily affected by the performance of the CH selection algorithm, which in turn is affected by the transmission range. A too small transmission range will cause failure of network connection and hence the CH selection algorithm. On the other hand, when the transmission range is set to $\sqrt{5}l$, the connectivity is guaranteed in the experiment and the CH selection algorithm can be reduced to selecting the node with the highest residual energy. Thus we evaluate the selection algorithm by varying the transmission range to change the connectivity. Here we assume the transmission power does not change with different transmission ranges to focus on the performance of the CH selection algorithm. In addition, since network lifetime could be as long as several hundreds of hours, we simplify the simulation such that in each CBS duration, we only simulate the network for 1 minute and use the obtained energy consumption to estimate the actual energy consumption for the whole duration. That is, if the CBS duration lasts for m minutes and a node consumes energy e in a minute, its total energy consumption for this duration is $m \cdot e$. We expect such approximation will not cause much difference on the results because of the fixed packet generation rate and hence the stable traffic pattern.

Fig. 4.7 shows the network lifetime under different transmission ranges. We use the box plot to better reveal its variation. For comparison, the dashed line represents the lifetime when only CBS is adopted and CHs cannot be changed. It can be seen that when the transmission range is short and the selection of CH to maintain connectivity is limited, the lifetime with scheduling switching is similar to or sometimes worse than the lifetime under CBS. In this case, transmissions in NCS consume more energy and lower the lifetime. However, when the transmission range slightly increases, its benefit becomes more evident with lifetime growing exponentially. When the range is above $1.72 \cdot l$, the lifetime reaches a maximum at about 720 hours, approximately 16 times of the lifetime under CBS. Notice that 16 is the number of nodes in the cluster closest to the sink, which consumes more energy than other clusters. This maximum lifetime demonstrates that the algorithm can efficiently and fairly dissipate the energy consumption among all possible nodes in the cluster to maximize network lifetime. On the other hand, the increasing network lifetime also indicates that the switching between CBS and NCS does not affect the connectivity of the network.

4.7 Conclusions

In this chapter, we have presented a hybrid scheme that integrates two communication scheduling approaches CBS and NCS to enable cluster-based WSNs to serve as network infrastructure of information collection in CPS. In CBS, a cycle based schedule for each CH is constructed based on the pre-determined routing tree. CBS minimizes the cycle length while maintaining the node order in the routing tree, which minimizes the intra-cluster collection delay and allows continuous packet forwarding from the source to the sink. In NCS, a CH-relay-member structure is proposed to replace the conventional CH-member structure. The introduction of relay nodes releases the CHs from the heavy burden of packet relaying so that the intra- and inter-cluster communications can be performed more efficiently. Our simulation results have shown

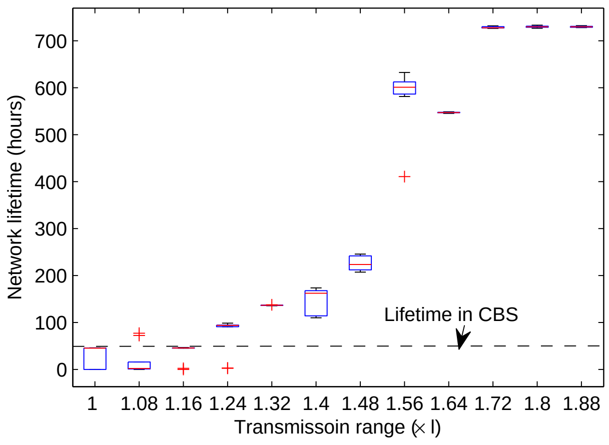


Figure 4.7: Box plot of network lifetime under different transmission ranges. The dashed line represents the network lifetime when CBS is adopted.

that the proposed approaches exhibit much better performance than existing scheduling approaches in terms of packet delay and throughput. The hybrid scheme integrates CBS and NCS without any interruption on data gathering during switching, allows the network to enjoy the benefits of both approaches to meet the stringent requirement for CPS.

Chapter 5

Conclusions

This dissertation focuses on the study of system evaluation and design for delay-sensitive wireless networks. At the evaluation part, we designed and implemented a general flexible network platform with hardware processing awareness to fill the gap between network simulations and the testbed implementation. The platform is constructed based on the co-simulation of the well-known network simulator NS-2 and the system-level hardware simulator SystemC. While both simulators are event-driven, we implemented the co-simulation by solving two critical issues: synchronization and communication. We also designed an easy interface to facilitate researchers to develop their own algorithms on the platform. To demonstrate the design motivation and the usage of the platform, we performed a case study on all-to-all broadcasting with network coding on this platform. We described the whole process of incorporating the processing modeling into the simulation. Extensive simulations revealed that hardware processing does not merely add the processing time to the packet delay. It will also intervene with the communication scheduling and cause more severe performance impact. With our platform, better algorithms can be designed by considering the interaction between communication and processing.

At the design part, we targeted at two important technologies - compression and clustering as widely used in wireless sensor networks. We first examined the compression effect on overall network performance in data collection using LZW as the example algorithm. Results showed that compression can have a long processing delay and its impact on the packet delay is dependent on the network traffic. Based on these observations, algorithms have been designed based on queueing theory to adaptively adjust compression strategy so that the compression benefit can be maximized. Second, we considered the communication synchronization problem caused by half-duplex antenna in cluster-based WSNs. We proposed both synchronous and asynchronous scheduling algorithms to efficiently schedule the intra- and inter-cluster communications within the network. We further combined two algorithms together to better support monitoring applications with both energy and delay requirements.

This dissertation is motivated by the increasing demands of delay-sensitive network applications and the observation that resource constraints in the network nodes can substantially affect the packet delay. The research we performed in this dissertation combines simulation methodology, hardware modeling, analysis and algorithm designs. The implemented platform can facilitate future researches on processing-intensive network applications while the designed algorithms can be applicable to a wide range of sensor applications with delay requirements. We expect our research to have a significant impact on design principles and system evaluation in delay-sensitive wireless networks.

Bibliography

[1] Inc. opnet technologies, opnet modeler, http://www.opnet.com/solutions/network_rd/modeler.html

[2] The network simulator: NS-2, http://www.isi.edu/nsnam/ns

[3] GloMoSim: Global mobile information systems simulation library, http://pcl.cs.ucla.edu/projects/glomosim/

[4] R. Ahlswede, N. Cai, S.-Y. Li and R. Yeung, "Network information flow," *IEEE Trans. Information Theory*, vol. 46, no. 4, pp. 1204-1216, 2000.

[5] M. Carson and D. Santay, "Nist net: A linux-based network emulation tool," *Computer Communication Review*, vol. 33, no. 3, pp. 111-126, 2003.

[6] L. Ni and P. Zheng, "Empower: A network emulator for wireline and wireless networks," *Proc. of IEEE INFOCOM*, 2003.

[7] S.-Y. Li, R. Yeung and N. Cai, "Linear network coding," *IEEE Trans. Information Theory*, vol. 49, no. 2, pp. 371-381, 2003.

[8] P. Chou, Y. Wu and K. Jain, "Practical network coding," *Proc. of the 51st Allerton Conf. Communication, Control and Computing*, Oct. 2003.

[9] K. Fall, "Network emulation in the VINT/NS simulator," *Proc. of the 4th IEEE Symposium on Computers and Communications*, 1999.

[10] Y. Wu, P.A. Chou and S.Y. Kung, "Information exchange in wireless networks with network coding and physical-layer broadcast," *Technical Report MSR-TR-2004-78, Microsoft Research (MSR),* Aug. 2004.

[11] C. Fragouli, J. Widmer and J.-Y. L. Boudec, "A network coding approach to energy efficient broadcasting: From theory to practice," *Proc. of IEEE INFOCOM*, 2006.

[12] F. Fummi, P. Gallo, S. Martini, G. Perbellini, M. Poncino and F. Ricciato, "A timing-accurate modeling and simulation environment for networked embedded systems," *Proc. of the 40th Design Automation Conference*, 2003.

[13] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan and D. Estrin, "Emstar: a software environment for developing and deploying wireless sensor networks," *Proc. of USENIX Technical Conference*, 2004.

[14] D. Herrscher and K. Rothermel, "A Dynamic Network Scenario Emulation Tool," *Proc. of the 11th International Conference on Computer Communications and Networks*, pp. 262-267, Miami, Oct. 2002.

[15] D. Johnson, T. Stack, R. Fish, D. Flickinger, L. Stoller, R. Ricci and J. Lepreau, "Mobile emulab: A robotic wireless and sensor network testbed," *Proc. of IEEE INFOCOM*, 2006.

[16] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Médard and J. Crowcroft, "XORs in the air: practical wireless network coding," *Proc. of ACM SIGCOMM*, 2006.

[17] L.E. Li, R. Ramjee, M. Buddhikot and S. Miller, "Network coding-based broadcast in mobile ad hoc networks," *Proc. of IEEE INFOCOM*, 2007.

[18] S. Chachulski, M. Jennings, S. Katti and D. Katabi, "Trading structure for randomness in wireless opportunistic routing,", *Proc. of SIGCOMM*, 2007.

[19] G. Werner-Allen, P. Swieskowski and M. Welsh, "Motelab: A wireless sensor network testbed," *Proc. of ISPN/SPOTS*, 2005.

[20] J. Schnerr, O. Bringmann, A. Viehl, and W. Rosenstiel. "High-performance timing simulation of embedded software," *DAC '08: Proc. of 45th Annual Design Automation Conference*, pp. 290-295, 2008.

[21] ModelNet, https://modelnet.sysnet.ucsd.edu/

[22] PlanetLab: an open platform for developing..., http://www.planet-lab.org/

[23] D. Raychaudhuri, I. Seskar, M. Ott, S. Ganu, K. Ramachandran, H. Kremo, R. Siracusa, H. Liu and M. Singh, "Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols," *Proc. of IEEE WCNC*, pp. 1664-1669, 2005.

[24] J. Schnerr, O. Bringmann, A. Viehl and W. Rosenstiel, "High-performance timing simulation of embedded software," *Proc. of Design Automation Conference*, pp.290-295, 2008.

[25] Z. Wang, A. Sanchez and A. Herkersdorf, "SciSim: a software performance estimation framework using source code instrumentation," *Proc. of the 7th International Workshop on Software and Performance*, 2008.

[26] E. Cheung, H. Hsieh and F. Balarin, "Framework for fast and accurate performance simulation of multiprocessor systems," *Proc. of the 2007 IEEE International High Level Design Validation and Test Workshop*, 2007.

[27] R. Le Moigne, O. Pasquier annd J-P. Calvez, "A generic RTOS model for real-time systems simulation with SystemC," *Proc. of International Conference on Design, Automation and Test in Europe*, 2004.

[28] W. Ye, J. Heidemann and D. Estrin. "Medium access control with coordinated adaptive sleeping for wireless sensor networks," *IEEE/ACM Trans. Networking*, vol. 12, no. 3, pp. 493-506, 2004.

[29] T. He, J. A. Stankovic, C. Lu and T. Abdelzaher, "SPEED: a stateless protocol for real-time communication in sensor networks," *Proc. of IEEE ICDCS*, 2003.

[30] E. Felemban, C. Lee and E. Ekici, "MMSPEED: multipath multi-SPEED protocol for qos guarantee of reliability and timeliness in wireless sensor networks," *IEEE Trans. Mobile Computing*, vol. 5, no. 6, pp. 738-754, 2006.

[31] T. He, B.M. Blum, J.A. Stankovic and T. Abdelzaher, "AIDA: adaptive application independent data aggregation in wireless sensor networks," *ACM Trans. Embedded Computing Systems*, 2003.

[32] S. Zhu, W. Wang and C.V. Ravishankar, "PERT: a new power-efficient real-time packet delivery scheme for sensor networks," *International Journal of Sensor Networks*, vol. 3, 2008.

[33] T.A. Welch, "A technique for high-performance data compression," *IEEE Computer*, vol. 17, no. 6, pp. 8-19, 1984.

[34] C.M. Sadler and M. Martonosi, "Data compression algorithms for energy-constrained devices in delay tolerant networks," *Proc. of SenSys*, 2006.

[35] K.C. Barr and K. Asanović, "Energy-aware lossless data compression," *ACM Trans. Computer Systems*, vol. 24, no. 3, pp. 250-291, 2006.

[36] TI MSP430F5418, http://focus.ti.com/docs/prod/folders/print/msp430f5418.html.

[37] X. Hong, M. Gerla, H. Wang and L. Clare, "Load balanced, energy-aware communications for mars sensor networks," *Proc. of the Aerospace Conference*, vol. 3, 2002.

[38] A. Jindal and K. Psounis, "Modeling spatially correlated data in sensor networks," *ACM Trans. Sensor Networks*, vol. 2, no. 4, pp. 466-499, 2006.

[39] I.F. Akyildiz, M.C. Vuran and O.B. Akan, "On exploiting spatial and temporal correlation in wireless sensor networks," *Proc. of WiOpt: Modelling and Optimization in Mobile, Ad hoc and Wireless Networks*, 2004.

[40] A. Scaglione and S. Servetto, "On the interdependence of routing and data compression in multi-hop sensor networks," *Wireless Networks*, vol. 11, pp. 149-160, 2005.

[41] S.J. Baek, G. de Veciana and X. Su, "Minimizing energy consumption in large-scale sensor networks through distributed data compression and hierarchical aggregation," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 6, pp. 1130-1140, 2004.

[42] S. Pattem, B. Krishnamachari and R. Govindan,"The impact of spatial correlation on routing with compression in wireless sensor networks," *ACM Trans. Sensor Networks*, vol. 4, no. 4, pp. 1-33, 2008.

[43] S.S. Pradhan, J. Kusuma and K. Ramchandran, "Distributed compression in a dense microsensor network," *IEEE Signal Processing Magazine*, vol. 19, no. 2, pp. 51-60, 2002.

[44] D. Slepian and J.K. Wolf, "Noiseless coding of correlated information sources", *IEEE Trans. Inform. Theory*, vol. 19, pp. 471-480, 1973.

[45] M.A. Sharaf, J. Beaver, A. Labrinidis and P.K. Chrysanthis, "TiNA: a scheme for temporal coherency-aware in-network aggregation," *Proc. Of the 3rd ACM international workshop on Data engineering for wireless and mobile access*, 2003.

[46] S. Yoon and C. Shahabi, "An energy conserving clustered aggregation technique leveraging spatial correlation," *Proc. of SECON*, 2004.

[47] T. Banerjee, K. Chowdhury and D.P. Agrawal, "Tree based data aggregation in sensor networks using polynomial regression," *Proc. of International Conference on Information Fusion*, 2005.

[48] T. Schoellhammer, B. Greenstein, B. Osterwiel, M. Wimbrow and D. Estrin, "Lightweight temporal compression of microclimate datasets wireless sensor networks," *Proc. of IEEE International Conference on Local Computer Networks*, 2004.

[49] E. Magli, M. Mancin and L. Merello, "Low-complexity video compression for wireless sensor networks," *Proc. of International Conference on Multimedia and Expo*, 2003.

[50] D. Petrovic,R.C. Shah, K. Ramchandran and J. Rabaey, "Data funneling: routing with aggregation and compression for wireless sensor networks," *Proc. of First IEEE International Workshop on Sensor Network Protocols and Applications*, 2003.

[51] T. Arici, B. Gedik, Y. Altunbasak and L. Liu, "PINCO: a pipelined in-network compression scheme for data collection in wireless sensor networks," *Proc. of ICCCN*, 2003.

[52] A. Reinhardt, M. Hollick and R. Steinmetz, "Stream-oriented lossless packet compression in wireless sensor networks," *Proc. of SECON*, 2009.

[53] N. Kimura and S. Latifi, "A survey on data compression in wireless sensor networks," *Proc. of ITCC*, 2005.

[54] M. Burrows and D.J. Wheeler, "A block-sorting lossless data compression algorithm," *Digital Systems Research Center Research Report*, 1994.

[55] IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications, *ISO/IEC 8802-11: 1999(E)*, 1999.

[56] G. Bianchi, "Performance analysis of the IEEE 802.11 distributed coordination function," *IEEE Journal in Selected Areas: Communication*, vol. 18, no. 3, pp. 535-547, 2000.

[57] E. Ziouva and T. Antonakopoulos, "CSMA/CA performance under high traffic conditions: throughput and delay analysis," *Computer Communications*, pp. 313C321, 2002.

[58] M.M. Carvalho and J.J. Garcia-Luna-Aceves, "Delay analysis of the IEEE802.11 in single-hop networks," *Proc. of ICNP*, 2003.

[59] O. Tickoo and B. Sikdar, "Queueing analysis and delay mitigation in IEEE 802.11 random access MAC based wireless networks," *Proc. of IEEE ICC*, 2004.

[60] H. Zhai, Y. Kwon and Y. Fang, "Performance analysis of IEEE 802.11 MAC protocols in wireless LANs," *Wireless Communications and Mobile Computing*, 2004.

[61] Y. Barowski, S. Biaz and P. Agrawal, "Towards the performance analysis of IEEE 802.11 in multi-hop Ad-Hoc networks," *Proc. of WCNC*, 2005.

[62] N. Bisnik and A.A. Abouzeid, "Queueing network models for delay analysis of multihop wireless Ad Hoc networks," *Ad Hoc Networks*, vol. 7, no. 1, pp. 79-97, 2009.

[63] Y. Wang, M.C. Vuran and S. Goddard, "Cross-layer analysis of the end-to-end delay distribution in wireless sensor networks," *Proc. of RTSS*, 2009.

[64] G. Bolch, S. Greiner, H. de Meer and K.S. Trivedi, *Queuing Networks and Markov Chains*, chapter 10, John Wiley and Sons, 1998.

[65] L. Sha, S. Gopalakrishnan, X. Liu and Q. Wang, "Cyber-physical systems: A new frontier," *Machine Learning in Cyber Trust*, pp. 3-13, Springer US, 2009.

[66] G. Gupta and M. Younis, "Load-balanced clustering in wireless sensor networks," *Proc. of IEEE ICC*, May 2003.

[67] W.R. Heinzelman, A. Chandrakasan and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," *Proc. of the 33rd Annual Hawaii International Conference on System Science*, 2000.

[68] O. Younis and S. Fahmy, "HEED: A hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks," *IEEE Trans. Mobile Computing*, vol. 3, no. 4, pp. 366-379, 2004.

[69] Z. Zhang, M. Ma and Y. Yang, "Energy efficient multi-hop polling in clusters of two-layered heterogeneous sensor networks," *IEEE Trans. Computers*, vol. 57, no. 2, pp. 231-245, 2008.

[70] A.A. Abbasi and M. Younis, "A survey on clustering algorithms for wireless sensor networks," *Computer Communications*, vol. 30, no. 14-15, pp. 2826-2841, 2007.

[71] M. Lotfinezhad, B. Liang and E.S. Sousa, "Adaptive cluster-based data collection in sensor networks with direct sink access," *IEEE Trans. Mobile Computing*, vol. 7, no. 7, pp. 884-897, 2008.

[72] H.J. Choe, P. Ghosh and S.K. Das, "QoS-aware data reporting control in cluster-based wireless sensor networks," *Computer Communications*, vol. 33, no. 11, pp. 1244-1254, 2010.

[73] L. Shi and A.O. Fapojuwo, "TDMA scheduling with optimized energy efficiency and minimum delay in clustered wireless sensor networks," *IEEE Trans. Mobile Computing*, vol. 9, no. 7, pp. 927-940, 2010.

[74] F. Cuomo, E. Cipollone and A. Abbagnale, "Performance analysis of IEEE 802.15.4 wireless sensor networks: An insight into the topology formation process," *Computer Networks*, 2009.

[75] W. Ye, J. Heidemann and D. Estrin, "An energy efficient MAC protocol for wireless sensor networks," *Proc. of IEEE INFOCOM*, 2002.

[76] T. van Dam and K. langendoen, "An adaptive energy-efficient MAC protocol for wireless sensor networks," *Proc. of ACM SenSys*, 2003.

[77] G. Lu, B. Krishnamachari and C.S. Raghavendra, "An adaptive energy-efficient and low-latency MAC for tree-based data gathering in sensor networks," *Wireless Communications & Mobile Computing* , vol. 7, no. 7, pp. 863-875, 2007.

[78] S. Liu, K. Fan and P. Sinha, "CMAC: an energy efficient MAC layer protocol using convergent packet forwarding for wireless sensor networks," *Proc. of IEEE SECON*, 2007.

[79] J. Kim, X. Lin, N.B. Shroff and P. Sinha, "On maximizing the lifetime of delay-sensitive wireless sensor networks with anycast," *Proc. of IEEE INFOCOM*, 2008.

[80] J. Kim, X. Lin and N.B. Shroff, "Optimal anycast technique for delay-sensitive energy-constrained asynchronous sensor networks," *IEEE/ACM Trans. Networking*, vol. 19, no. 2, pp. 484-497, 2011.

[81] P.A.W. Lewis and G.S. Shedler. "Simulation of nonhomogenous Poisson processes with log-linear rate function," *Biometrika*, vol. 63, no. 3, pp. 501-505, 1976.

[82] S.C. Ergen and V. Pravin. "TDMA scheduling algorithms for wireless sensor networks," *Wireless Networks*, vol. 16, no. 4, pp. 985-997, 2010.

[83] I. Rhee, A. Warrier, M. Aia, J. Min and M.L. Sichitiu, “Z-MAC: a hybrid MAC for wireless sensor networks,” *IEEE/ACM Transactions on Networking*, vol. 16, no. 3, pp. 511-524.

[84] G.S. Ahn, S.G. Hong, E. Miluzzo, A.T. Campbell and F. Cuomo, “Funneling-MAC: a localized, sink-oriented MAC for boosting fidelity in sensor networks,” *Proc. of Sensys*, 2006.

[85] J. Chou, D. Petrovic and K. Ramchandran, “A distributed and adaptive signal processing approach to reducing energy consumption in sensor networks,” *Proc. of IEEE INFOCOM*, 2003.

[86] M. Yavis, N. K ushalnagar, H. Singh, A. Rangarajan, Y. Liu and S. Singh, “Exploiting heterogeneity in sensor netw orks,” *Proc. of IEEE INFOCOM*, 2005.

[87] X. Deng and Y. Yang, “Cluster communication synchronization in delay-sensitive wireless sensor networks,” *Proc. of IEEE DCOSS*, 2013.

[88] X. Deng and Y. Yang, “An efficient MAC multicast protocol for reliable wireless communications with network coding,” *Proc. of IEEE GLOBECOM*, 2011.

[89] X. Deng and Y. Yang, “On-line adaptive compression in delay sensitive wireless sensor networks,” *Proc. of IEEE MASS*, 2010.

[90] X. Deng, Y. Yang and S. Hong, “A flexible platform for hardware-aware network experiments and a case study on wireless network coding,” *Proc. of IEEE INFOCOM*, 2010.

[91] X. Deng, Y. Yang and S. Hong, “A flexible platform for hardware-aware network experiments and a case study on wireless network coding,” *IEEE Trans. Networks*, vol. 21, no. 1, pp. 146-161, 2012.

[92] X. Deng and Y. Yang, “On-line adaptive compression in delay sensitive wireless sensor networks,” *IEEE Trans. Computers*, vol. 61, no. 10, pp. 1429-1442, 2011.

[93] X. Deng and Y. Yang, “Adopting Compression in Wireless Sensor Networks,” *Scalable Computing and Communications: Theory and Practice*, Wiley-IEEE Computer Society Press, New Jersey, USA, 2013.

[94] X. Deng and Y. Yang, “Communication Synchronization in Cluster-Based Sensor Networks for Cyber-Physical Systems,” *accepted in IEEE Trans. Emerging Topics in Computing*, 2013.