AN EXPLORATION OF MECHANISMS AND POLICIES FOR GATEWAYS IN REAL-TIME EMBEDDED SYSTEMS

Submitted in partial fulfillment of the requirements for

the degree of

Doctor of Philosophy

in

Electrical and Computer Engineering

Justin David Ray

B.S., Electrical Engineering, Texas A&M University

Carnegie Mellon University

Pittsburgh, Pennsylvania

December 2013

Call unto me, and I will answer thee,
and show thee great and mighty things, which thou knowest not.
Jeremiah 33:3

Abstract

It is becoming increasingly common to connect traditional embedded system networks to the Internet for remote monitoring, high-level control, and integration. One architectural approach to building internetworked systems is to add a gateway between the embedded system and the external network. These gateways must transfer data between two heterogeneous networks without inducing application failures due to variations in timing or bandwidth between the two networks. Despite the importance of gateways, there is no clear recipe for designing them. To study gateway design, we examine gateway mechanisms that can be used to handle data passing through a gateway and describe policies for configuring these mechanisms. In this work, we examine the differences between generic mechanisms (i.e. queues) and application-aware mechanisms that use knowledge of the data being transferred. Using simulation with abstract network models, we compare the performance of these mechanisms and show that application-aware mechanisms can be useful in improving gateway performance in some situations. We also use a case study of a traffic control application to evaluate the performance of gateway mechanisms with simulations that model different network and environmental scenarios. We find that selection of the proper gateway mechanism can improve performance of the traffic control application, and we provide selection guidance based on the mean inter-arrival time of the network. These results show that in most scenarios, application-aware filter mechanisms outperform generic queue mechanisms.

Acknowledgments

This work is dedicated to my dear wife, Starry. Without her love and support, none of this would be possible.

I would like to thank my advisor, Dr. Philip Koopman, for providing me with the advice, guidance, and experience that has brought me this far. I would like to thank my committee, Dr. Hyong Kim, Dr. John Lehoczky, and Dr. Anthony Rowe, for their insight and patience.

I would like to thank my parents, Terry and Patricia Ray, my parents-in-law, Glenn and Shirley Kennedy, and my sister Ashley Pèrez for their constant encouragement. I would like to thank Cleve Cooke and Amy Wung Tsao for notes of wisdom and prayers when they were needed most. I would like to thank Jen Black for paving the way. I would like to thank Michael Wagner for time every time it was needed.

I would like to thank OPNET Technologies, Inc. for their generous donation of the licenses for OPNET® Modeler and the wireless packages which made the traffic simulation evaluations possible. I would like to thank Enthought, Inc. for academic licenses to the Enthought Canopy Python tools which contributed significantly to the analysis of simulation results.

This research was supported in part by GM-Carnegie Mellon Vehicular Information Technology Collaborative Research Lab, Honeywell, and Bombardier Transportation.

Table of Contents

List of Tables

List of Figures

List of Acronyms

ACK: denotes a TCP acknowledge packet
AODV: Ad hoc On-Demand Distance Vector
API: Application Programming Interface
CA: Cellular Automata
CAN: Controller Area Network
CSMA: Carrier-Sense Multiple Access
DES: Discrete Event Simulation
DLE: Decaying Linear Extrapolation
EFC: Energy-based Fuel Consumption
FIFO: First-In-First-Out
FPGA: Field Programmable Gate Array
FTP: File Transfer Protocol
IDA: Independent Delay Analysis
IEEE: Institute of Electrical and Electronics Engineers
IP: Internet Protocol
HSUPA: High-Speed Uplink Packet Access
LAN: Local Area Network
MANET: Mobile Ad-hoc Network
Mbps: Megabit per second
MSE: Mean Squared Error
NFR: Normalized Fuel Rate
OBD-II: On-Board Diagnostic II
PDF: Probability Distribution Function
RED: Random Early Detect
RMA: Rate Monotonic Analysis
RMSE: Root Mean Squared Error
RN: Receiving Network
RTT: Round Trip Time
TCP: Transmission Control Protocol
TDMA: Time Division Multiple Access

TN: Transmitting Network
TTP: Time Triggered Protocol
UDP: User Datagram Protocol
VM: Virtual Machine
VSP: Vehicle Specific Power

List of Symbols

α	Idle fuel rate in mL/s (EFC model)
β_1	Energy efficiency in ml/kJ (EFC model)
β_2	Energy-acceleration efficiency in $\frac{mL}{kJ \cdot m/s^2}$ (EFC model)
β_s	coefficient that relates VSP and NFR (NFR model)
δ	fixed delay value (Independent Delay Analysis)
δ_{tx}	the transmission period for the guidance messages (Internet delay model)
$\hat{S}_\delta$	Sequence of estimated values with fixed delay δ (Independent Delay Analysis)
$\hat{s}_{\delta,i}$	the i^{th} estimated sample with fixed delay δ (Independent Delay Analysis)
$\hat{s}_{d,i}$	the i^{th} estimated sample with delay d (Filter Underflow Policies)
μ	the processing rate of the queue, in bits/s (Internet delay model)
ρ	the overall density of traffic in the simulation (CA traffic model)
VSP	vehicle specific power, a measure of vehicle power demand (NFR model)
a	vehicle acceleration (NFR & EFC model)
a_{max}	the maximum positive acceleration (cells/tick/tick) allowed for vehicles in the simulation (CA traffic model)
B	a random process that determines the amount of network traffic injected between arriving probe packets (Internet delay model)
b_1	Drag force in kN, mainly related to rolling drag (EFC model)
b_2	Drag force in $\frac{kN}{m/s^2}$, mainly related to aerodynamic resistance (EFC model)
D	fixed packet delay in s (Internet delay model)

D_0 the position (cell number) of the stoplight (CA traffic model)

d_i the number of empty cells between the current vehicle and the following vehicle (CA traffic model)

E_F complete set of error metrics for a data model F (Independent Delay Analysis)

e_δ mean squared error between S and $\hat{S}_\delta$ (Independent Delay Analysis)

L the length of the ring road in cells (CA traffic model)

l the size of the vehicle in cells (CA traffic model)

M mass in kg (EFC model)

N the number of vehicles in the simulation (CA traffic model)

N_0 the number of vehicles where $v_i = 0$ (CA traffic model)

P the packet size

p the probability of random deceleration in the CA model (CA traffic model)

r_i the number of cells to the stoplight (CA traffic model)

R_t the total tractive force (EFC model)

S input data set (Independent Delay Analysis)

s_i i^{th} sample of the data set S (Independent Delay Analysis)

t the current time (CA traffic model)

v vehicle velocity (NFR & EFC model)

v_i^d the current desired velocity (cells/s) computed by the guidance algorithm and transmitted to each vehicle (CA traffic model)

v_i the current velocity (cells/s) of the i^{th} vehicle (CA traffic model)

v_t a candidate desired velocity value (traffic guidance algorithm)

v_{max} the maximum velocity (cells/tick) allowed for vehicles in the simulation (CA traffic model)

x_i the current position (cell number) of the rear of the i^{th} vehicle (CA traffic model)

Chapter 1

Introduction

Embedded systems, which traditionally have operated in isolation or on closed networks, are being connected to the Internet or to other networked systems to increase functionality and consolidate operations. Although these connections initially tend to be non-real-time, once these internetworked systems are in place, the trend toward greater integration is going to encompass real-time and safety-critical applications. For example, the OnStar system produced by General Motors initially provided emergency assistance, remote diagnostics, the ability to unlock car doors remotely, and activate the vehicle's horns and lights (while the vehicle is stopped). Later versions of the system include the ability to disable a stolen vehicle remotely by reducing its maximum speed [1]. Other automotive manufacturers offer similar systems. BMW ConnectedDrive offers in-car Internet access [2]. Ford's SYNC system can download real-time traffic and weather information, as well as provide remote vehicle diagnostics [3]. [4] describes the popularity and growth of telematics systems, noting a survey in which 70% of telematics users would require a similar system in their next vehicle purchase. A proper understanding of the issues arising in gateway design is important as the popularity of and demand for these systems continues to grow.

1.1 Problem and scope

This research addresses the problem of using gateways to connect enterprise systems to embedded, real-time systems. The primary research questions are:

- What mechanisms can be used to handle data in a gateway, and what policies can be used to configure those mechanisms?
- Can application-aware mechanisms be used to improve gateway performance compared to generic mechanisms?
- How do various gateway mechanisms and policies perform when studied in simulation models?

The problem scope is defined with respect to existing routing mechanisms and security approaches. We evaluate these approaches to determine which can be applied to real-time embedded systems.

1.1.1 Routers and Gateways

Routers and gateways in the Internet domain, which perform a function similar to an embedded gateway, use queues to manage flows of information. Historically, routers were limited to making routing decisions based only on the packet headers, but deep packet inspection is being used to provide more sophisticated network security [5]. This trend suggests that gateways can manage the flow of information more effectively if they are aware of the type of the information they are processing. Application awareness allows the use of mechanisms that filter, aggregate, and prioritize messages based on message data and not just message source and arrival order. Application-aware approaches are already being used to manage streaming video services [6]. Using tuned mechanisms can improve

real-time behavior, since gateway mechanisms can be selected to meet specific timing requirements. Application-aware approaches also allow the system to be more survivable since applications can clearly define the failure semantics of the data streams being used.

1.1.2 Security Approaches

Since the networks on either side of the gateway have different timing properties, normal (non-faulty) timing of message arrival on one side may cause the gateway to fail to meet timing requirements on the other side. Standard security and survivability practices, while important, are not geared toward protecting systems against these faults, nor do they address timing issues that are introduced by the interaction between two types of networks. Enterprise network protection techniques, such as firewalls and intrusion detection systems, are focused on restricting information flow. While practical implementations of embedded gateways may also utilize some traditional enterprise network protection techniques, they need to go beyond these techniques to proactively manage the timing of information passing through the gateway.

1.2 Embedded Gateways

As a working definition, an embedded gateway is defined as *the physical devices, software, and application logic required to bridge communication between two networks and allow nodes on one network to transmit information to nodes on another network within the constraints of the two networks*. In most practical implementations, a gateway will be a single physical device with two network interfaces that contain all the required software and information needed to coordinate information flow. Even in the case of a system where some of the gateway functionality is allocated to additional nodes in the network, the data

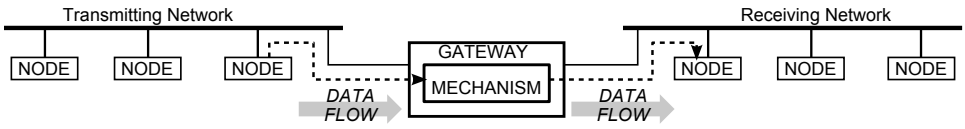


Figure 1.1. Flow of data in an embedded gateway system

flows should be similar, so the mechanism-selection guidance provided here still should be applicable.

Figure 1.1 shows an example system with two networks connected by a gateway. This figure also shows the convention that is used throughout this document. The Transmitting Network (TN) is the network that contains the node(s) sending data to the gateway. The Receiving Network (RN) is the network that contains the node(s) receiving data from the gateway. The figure also shows the gateway mechanism. A gateway mechanism (hereafter referred to simply as a mechanism) is a combination of message processing logic and storage intended to manage a particular flow of data between two networks.

A great deal of this work is concerned with evaluating gateway mechanisms. A **gateway mechanism** is defined here as *a set of data structures and algorithms that handles a flow of data through the gateway*. Usually, the mechanism will be implemented in software on a general-purpose computing platform, although FPGA or custom chip implementations are also feasible as the technology and understanding of gateway mechanisms matures. Some mechanisms, such as queues, are existing mechanisms used in related applications. Other mechanisms, such as filters, are new mechanisms that we have developed to address the issues arising from the study of queues in real-time embedded gateway applications. A **gateway policy** is defined as *the algorithm selection and choice of parameters needed to fully define the function of an instantiation of a gateway mechanism*.

In many applications, data likely will be sent in both directions through the gateway, and there may be more than one data type being sent. In our evaluations, we consider the case of a single data stream going from the TN to the RN. The single-stream case can be

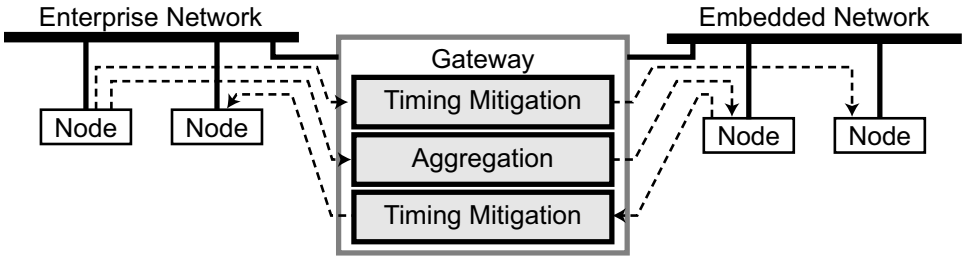


Figure 1.2. Gateway with multiple data streams and mechanisms

expanded to cover more complex data flows by assuming (a) that each data stream has its own mechanisms and appropriate message time slots on both networks and (b) that the system is provisioned in such a way that there is enough available memory and processing power within the gateway to handle all the required streams. Figure 1.2 shows an example gateway system with three data streams, each handled by its own mechanism in the gateway. In this figure, the enterprise network is the TN for the first two streams and the RN for the third stream (and vice-versa for the embedded network). The traffic for the three streams is assumed to be independent, so each stream can be analyzed separately. To be independent, we assume that effects of other data streams are included in the aggregate behavior of the arrival characteristics of the network.

One limitation to this approach is that, without bidirectional mechanisms, the gateway cannot implement explicit, end-to-end acknowledgment mechanisms that span both networks. We choose not to consider this case because multi-hop acknowledgments are not common in embedded systems. Furthermore, for the common case of broadcast networks, there is currently no standard approach that can be studied. In a non-real-time system, an end-to-end acknowledgment mechanism can be implemented with two one-way mechanisms.

1.2.1 Gateway Scenarios

There are several scenarios in which a gateway can be a useful architecture. The requirements for the gateway will likely be different in each scenario and further vary, depending on the needs of a particular application. A gateway in a real system might be responsible for several data flows that could encompass more than one of these scenarios.

- *Enterprise-to-Embedded:* data is sent to the gateway over enterprise networks (such as the Internet or a LAN). The gateway then sends these data out on a real-time embedded network. An example of this system is a supervisory control application that runs on a corporate LAN and connects via gateway to a factory control-network.

- *Embedded-to-Enterprise:* the gateway receives data from a real-time embedded network and then sends it to a server or personal computer over an enterprise network. This scenario might arise if an embedded system (such as a thermostat on a home automation network) reports its status to an Internet server.

- *Embedded-to-Embedded:* the gateway connects two embedded networks that are of a different type. The two networks could also be of the same type but configured to run at different speeds or with different schedules. For example, many automobiles have several embedded networks. These networks are interconnected to implement system features, such as when an airbag LIN system is connected to the OnStar system.

In this research, we focus on the enterprise-to-embedded scenario for applications that use real-time, numeric-valued data. Within this scope, we evaluate options for selecting which mechanisms to use and the policies needed to configure them.

1.3 An Approach for Evaluating Gateway Mechanisms

In order to evaluate mechanisms, we use two different simulation techniques to measure the performance of the gateway as data flows through it. Because the performance of gateway mechanisms is so heavily dependent on the characteristics of the application, we make no attempt to provide general solutions to the selection problem. Instead, we present insights, simulation techniques, and analysis methods that provide a workflow for evaluating gateway mechanisms in the context of a specific application.

The first technique is a simulation approach based on abstract network models. This approach is described in more detail in Chapter 3. Networks in the system are modeled by random processes: (a) an arrival process that models the enterprise network and feeds data into a gateway mechanism, and (b) a service process that models embedded networks and extracts data from the mechanism. The characteristics of the networks are represented in a general way by the probability distributions used for these processes. These models are far removed from the reality of physical networks and protocols, thus the name Abstract Network Simulation, but they have the advantage of providing insights while being relatively simple to implement and execute. Abstract network models have few parameters (e.g. distribution parameters for the arrival and service process), as opposed to the full network simulations such as OPNET® Modeler [7] which have dozens of configurable parameters for every node in the system. When we run simulations using these abstract models, we use input data collected while driving a vehicle in several real-world traffic scenarios. These data were chosen because they represent reasonable values (a range of vehicle speeds) that would be representative of real-time data used, for example, in a traffic control scenario. Although the simulation results provide insight into the way mechanisms interact with data, the results are necessarily specific to the data used. The same simulation approach can be

applied to another, similar application to obtain a comparison of mechanism performance that is applicable to that application.

The second technique is a case study that uses the OPNET® Modeler simulation framework [7] to model a simple traffic control scenario, as described in Chapter 7. The traffic control algorithm used in the case study is published in [8]. The traffic simulation is implemented using cellular automata, and the network simulation framework and models are provided by OPNET®. The case study has the advantage of being a realistic application with real-world metrics that can be used to compare performance. However, there are also many more parameters that can affect the outcome of the simulation. We have chosen simulation experiments with parameters that represent realistic conditions likely to be encountered in the traffic environment. As with the abstract network simulations, the results obtained from these simulations are specific to this application. The analysis of the case study results provides a road map for applying this technique to other applications.

1.4 Research Contributions

In this research, we study the performance of various gateway mechanisms and policies for systems that conform to the enterprise-to-embedded scenario and use real-time, periodic, numeric-valued data. Within this context, this research makes the following contributions:

- **This research identifies mechanisms and policies that can be used to mitigate problems that arise in a gateway.**

  In order to design a gateway, we must first know what mechanisms are available for use as data handling methods. Queue mechanisms are widely used in routing applications, which are similar to gateway applications, except that they do not deal with the real-time requirements of embedded systems. In this research, we consolidate

existing policies that have been developed for queue mechanisms and evaluate them for usefulness in enterprise-to-embedded gateway scenarios. But the ways in which queue mechanisms can be tuned are somewhat limited, and the simulation results from the case study show that adjusting these parameters does not affect performance. So we introduce a new class of mechanisms called filters that can address shortcomings identified in queue mechanisms and improve performance in some cases. Chapter 4 presents the mechanisms that are examined in this research.

- **This research demonstrates that application-aware mechanisms exhibit improved performance when compared with generic mechanisms in the enterprise-to-embedded gateway applications studied.**

  Generic mechanisms (e.g. what most Internet routers use to manage packet flows) make decisions based on source and destination routes and based on the arrival timing of data at the router. We compare this approach to application-aware mechanisms and show, that by providing a gateway with information about the semantic content of messages, a mechanism can improve performance of the application using the gateway over generic mechanisms. The application-aware mechanisms studied here differ from many Internet approaches to meeting application-specific performance requirements. Some approaches implement active elements in the network (that could be considered gateway mechanisms), but their application is usually focused on congestion management (e.g. [9]). Our approach differs because the application-aware mechanisms we propose can actually modify or synthesize messages, but provide no capability to modify the characteristics of the network or traffic sources. This kind of mechanism would not be suitable for a streaming video application, but would be suitable for control applications commonly seen in embedded systems.

To address this contribution, we show that application-aware mechanisms can improve gateway performance when compared to generic mechanisms. In Chapter 3, we present a framework for evaluating mechanisms using abstract network models. The results of this evaluation in Chapter 6 show that, for the vehicle speed data we studied, the mean error of data in the gateway is reduced when application-aware mechanisms are employed. Furthermore, the results presented in Chapter 7 compare generic and application-aware mechanisms in a traffic control case study and show that the application-aware mechanisms improve overall vehicle performance and fuel economy in some cases.

- **This research uses simulation evaluation results to give guidance for selecting the appropriate mechanism and policy for a particular scenario.**

  Understanding how to choose a mechanism for a particular application is necessary in order to adopt gateway architectures. In Chapter 5, we show how mechanisms can be analyzed independent of network characteristics to provide insight into selecting mechanism policies. Furthermore, we are able to show that there are cases where filter mechanisms provide improved performance in the traffic case study. In Chapter 8, we provide a decision rule for the traffic case study that selects between application-aware and generic mechanisms based on the inter-arrival characteristics of the enterprise network.

The remainder of this research is organized as follows: Chapters 1 through 2 introduce the work. Chapter 1 provides an introduction to the problem area and a description of the goals of the research. Chapter 2 provides background on concepts related to the problem and addresses work in related areas that bears on gateway design.

Chapters 3 through 6 describe how to evaluate gateways using abstract network models and provide candidate mechanisms based on insights gained from this work. Some portions of these chapters were published in [10]. Chapter 3 describes a method for simulating gateway systems using abstract network models. Chapter 4 describes two classes of gateway mechanisms, including the filter mechanism, a new mechanism we propose to address problems with queue mechanisms. This chapter also defines policies that can be used to configure the mechanisms. It is these mechanisms and policies that are studied in the abstract network model simulations and, later, in the traffic control case study. Chapter 5 presents Independent Delay Analysis, an analysis method that provides insights into how network and data characteristics affect the error performance of various mechanisms. A key insight here is that the Independent Delay Analysis can be conducted using only a numeric analysis tool such as Matlab, independent of any network model or simulation. The results then can be used to obtain insights about mechanism parameters and mechanism selection for a wide range of networks. Chapter 6 presents the results of simulations using abstract network models.

Chapter 7 describes a case study that evaluates gateway mechanisms using a traffic control application with realistic network models provided by OPNET®. This chapter also presents the results of the simulations. Chapter 8 describes selection rules for choosing gateway mechanisms based on the traffic control application and discusses the relationship of these results to the characteristics of the systems being simulated. Chapter 9 presents conclusions and summarizes the contributions of this research.

Chapter 2

Background and Related Work

2.1 Overview

While the idea of a gateway (a device that connects or bridges two networks) is almost as old as networks themselves [11], many gateways (such as Internet routers) deal with connections between networks of the same type. Internet routers are typically designed to optimize throughput and provide fair delivery during congestion, and this is primarily done with queues and queue management techniques [12, 13], but these approaches are not concerned with real-time delivery.

Various aspects of embedded system gateways are also being studied, although the primary focus of work being done in this area (including [14, 15, 16]) is on the implementation of the gateway device and protocol translation. [14] describes an Internet-connected Controller Area Network (CAN) gateway, which is an example of the enterprise-to-embedded scenario. Two examples of embedded-to-embedded gateways are [15], which describes a gateway between FlexRay and CAN networks, with a focus on the reliability of the gateway device, and [16], which describes a FlexRay-to-CAN network. In contrast to this latter work, we focus on the enterprise-to-embedded scenario (see Section 1.2.1). Also, all of these gateway implementations are primarily focused on protocol translation, while our

work is focused on issues of gateway design as they relate to data delivery and application performance.

[17] describes an adaptive, multi-hop routing protocol that uses dual-homed wireless devices as gateways for heterogeneous wireless networks. Although this work does deal with heterogeneous networks, it is focused on routing messages through a series of wireless networks, while our work deals with the timing aspects of data delivery. However, it is relevant to our work because a detailed study of the delays introduced by a multi-hop routing protocol could be used to provide more sophisticated models for the enterprise network in the enterprise-to-embedded scenario.

The remainder of this chapter describes several related areas of study that contribute techniques and ideas useful to the selection and design of embedded gateway mechanisms.

2.2 Mechanism and Policy Separation

The notion of mechanism and policy separation is advanced by [18] in the context of system resource allocations. That work provides several important insights that are directly relevant to the development of mechanisms for embedded gateways. First, mechanism and policy separation allows mechanisms to be formulated ahead of time so that policies can be constructed as needed by applications. For embedded gateways, the mechanisms identified and studied in this work provide a basis for formulating policies to meet application requirements. Second, the authors note that not all policies can be implemented with a given set of mechanisms. That insight holds for gateway applications as well. There are some applications which impose requirements that cannot be met with existing mechanisms. Even in this case, it is still useful to know when previously-studied mechanisms are inadequate. Reaching this conclusion makes plain the need for changes in system architecture (or the development of new mechanisms) to meet requirements. [19] addresses the idea of mech-

anism and policy separation for security and notes that mechanisms may be used to fulfill multiple policies.

The mechanisms used in our work are gateway mechanisms that manage the flow of data between networks. For our purposes, mechanisms are (a) data structures for storing message data and (b) rules and algorithms for manipulating those structures. Mechanisms may have configurable aspects, such as threshold parameters or algorithm selection. Policies are the particular instantiation of a mechanism with parameters. Our work also provides policy guidance in the form of selection criteria and analysis methods, which can be used to select and tune mechanisms to meet specific application requirements.

2.3 Types of Networks

There are many different networks which might be connected to a gateway. Each network may have a different approach to scheduling, framing, and access control. This section briefly describes the types of networks commonly used and some of their distinguishing properties.

2.3.1 Enterprise Networks

For non-real-time networks (referred to here as **enterprise networks**), the most common network type is a packet-switched IP network. An enterprise network could be as simple as an Ethernet-based local area network, or it may encompass the entire Internet. At a local level, a great deal of congestion control and reliability is possible through the use of switched networks, but the Internet is a multi-tiered system where different tiers are typically controlled by different entities. Shared resources and dynamic routing protocols make it difficult to predict packet latency, which can be highly variable over time [20].

Wireless networks can also be used in an enterprise context. This context includes wireless local area network protocols such as IEEE 802.11n [21] and cellular data protocols such as HSUPA [22], which are used with digital smart phones. Typically, wireless links are the last hop on a packet's journey through a larger network, so we consider the behavior of the wireless data link to be subsumed by the larger effects of the entire enterprise network. In order to evaluate systems that include enterprise networks, we use models that capture the timing and bandwidth characteristics. Typically, data on enterprise networks arrives at random time intervals because of varying factors such as protocol startup, multiple hops between routers, and congestion from other network traffic. A Poisson process is used to characterize this behavior [23]. Other properties relevant to the model are high bandwidth (relative to embedded networks) and the ability to send large packets (greater than 1 MB).

2.3.2 Embedded Networks

We use the term **embedded networks** to describe networks with real-time properties typically used in embedded applications such as cars, copy machines, and elevators. Embedded networks are usually multicast networks which use a shared bus. Some networks may use multiple buses or star configurations for increased reliability. Although there are other differences in scheduling and framing, the descriptions below classify embedded networks by their access control mechanisms.

FlexRAY [24] is an embedded network that uses Time Division Multiple Access (TDMA) to arbitrate media access. It provides reliable, real-time delivery of periodic messages. FlexRAY has a message schedule with an option for static and dynamic segments. In the static segment, an entire time slot is allocated for each message in the schedule. In the dynamic segment, mini-slots are used to determine priority for sending optional messages

(a variation of Reservation CSMA). Timely delivery of messages in the dynamic segment is not guaranteed.

The Time-Triggered Protocol (TTP) [25] is another embedded network that uses TDMA. TTP uses a static message schedule similar to the static segment of FlexRAY. TTP also provides a group membership protocol that allows network nodes to reach agreement on the value of messages being transmitted. The Controller Area Network (CAN) protocol [26] uses binary countdown (based on the message ID) to arbitrate access to the bus. Binary countdown allows efficient network utilization with no pre-defined schedule, but limits data rates because of bit propagation delay. Rate Monotonic Analysis can be used to develop a static priority schedule for CAN networks that ensures that all senders can meet periodic deadlines. TTCAN [27] is a modification of the CAN protocol that provides additional guarantees on latency to improve time-triggered operation.

When modeling embedded networks, the most important feature distinguishing embedded networks from enterprise networks is that embedded networks support real-time deadlines. In the case of TDMA networks, the message schedule is fixed, and the sender has a specific timeslot in which to send the message. CAN networks are slightly more flexible, but still must transmit within their deadline if RMA scheduling is used. Although the TDMA networks can achieve higher data rates (20 Mbps) and have relatively large maximum message sizes ($>$1 kilobyte) compared to CAN (1 Mbps, 8 byte messages), the overall available bandwidth is much lower for many of these networks than for enterprise networks.

2.4 Event-Triggered and Time-Triggered Architectures

When selecting mechanisms for use in a gateway, the system architecture can have a significant effect on the way data is handled by the system and, thus, how the data must be

handled by the gateway. Event-triggered and time-triggered systems are two common time-management architectures used in real-time systems. Each architecture is described here in the context of the data flowing through the gateway. A more thorough discussion can be found in [28].

In event-triggered systems, nodes in the system take action based on state changes in the system or in the external environment. The system takes appropriate action to respond to events. For a distributed system, data is sent only when an event occurs, which means that there are fewer messages in the system, but each message is more important because it represents a change upon which the system must act. If data crosses the gateway, it is important that the gateway preserve the semantics (and, possibly, the ordering) of events to maintain consistency between the TN and RN parts of the system.

Time-triggered systems perform a regular set of tasks based on a current view of the system state. These tasks are performed whether or not the state of the system or the environment has changed. In a distributed system, time-triggered architectures are typically implemented by broadcasting periodic state messages to other nodes in the system. If this periodic update message is sent through the gateway, then it is more important to preserve an accurate notion of current state than it is for every message to be delivered. There is also a concern if one of the networks is bandwidth constrained. In a bandwidth-constrained case, it may be desirable for the gateway to filter the time-triggered state messages to reduce outgoing bandwidth.

Given the different underlying assumptions for time-triggered and event-triggered systems, it is likely that gateways will need to manage the data flows differently (e.g., use different mechanisms) for each architecture. In this work, we focus on applications with a time-triggered architecture.

2.5 Queuing Theory and Queue Management

Work on queuing theory was first published in 1909 by A.K. Erlang (as noted in [29]), and much work has been done since then to develop models for various queue scenarios. For gateways, standard queuing notation can be used to describe the gateway scenarios in general. For example, the enterprise-to-embedded scenario (see Section 1.2.1) corresponds to an M/D/1 queue, where a Markov arrival process (the enterprise TN) feeds data into the queue and a deterministic service process (periodic messages on the embedded RN) remove data from it. Real-time queuing theory was addressed in [30] by applying queuing theory to scheduling of packets in a packet-switched network to analyze the system for missed deadlines.

The difficulty in applying queuing theory to the gateway problem is that its models are restrictive and cannot model many real-world mechanisms. They also give little insight into metrics relevant to application-aware mechanisms, such as error rates (which depend on message values as well as arrival timing). Chapter 6 shows that application-aware mechanisms can be more effective than queues at managing the data flow, so an approach that allows us to include these mechanisms is needed. Although queues likely will prove useful in other gateway scenarios, the simulation approach enables us to study a greater variety of mechanisms.

Queuing is the primary mechanism used to manage packets in Internet routers, and much work has been done with active queue management techniques for optimizing throughput and implementing congestion control algorithms. Random Early Detect (RED) [13] or Blue [12] are two examples that are designed to maximize throughput and reduce congestion. These techniques are usually applied to unbounded queues. [31] modifies RED in an attempt to maintain a target queue length to constrain memory requirements in the router. These approaches are related to the drop polices applied to the bounded queues, but, as

discussed in Chapter 6, both bounding queue length and applying different drop policies are of limited effectiveness for reducing error for an embedded gateway in a time-triggered, real-time application scenario.

2.6 Discrete Event Simulation

All of the simulation techniques used in this research are based on discrete event simulation (DES). In DES, an ordered sequence of events is executed, and each event modifies the state of the system model. These modifications may include the spawning of additional events. DES is an approach that is commonly used to model systems in research applications. For example, [32] describes the use of DES for evaluating a vehicle stability application implemented on an embedded network.

We use a custom-built DES to model abstract network models (see Chapter 3). We also use OPNET® Modeler [7], a commercial application, to implement the traffic and network simulations for the case study in Chapter 7.

2.7 Traffic Flow Modeling

There is a substantial body of work devoted to the subject area of modeling and analyzing traffic flow. For an excellent overview, the reader is referred to *An Introduction to the Theory of Traffic Flow*[33].

Our approach to modeling traffic flow is based on the application in [8]. This application uses a cellular automata model originally described in [34]. This approach to simulating traffic flow is known as **microsimulation**. The alternative, **macrosimulation**, models flows of traffic instead of individual vehicles.

The microsimulation approach is best suited for this work because it provides a way to model each vehicle in the system individually so that each car can receive and act on guidance provided by the central traffic controller. The application in [8] was chosen for the case study because it provides an application that is feasible to simulate in the OPNET® Modeler simulation environment to take advantage of the network models and simulation capabilities that Modeler provides. Studying the application provides a variety of scenario parameters that can be varied while comparing the effectiveness of the gateway mechanisms. A full explanation of the implementation details can be found in Chapter 7.

2.8 Summary

We frame the problem of embedded gateway design in terms of mechanisms and policies which can be used to manage the flow of data. Mechanisms are (a) data structures for storing message data and (b) rules and algorithms for manipulating those structures. Mechanisms may have configurable aspects, such as threshold parameters or algorithm selection. Policies are the particular instantiation of a mechanism with parameters. Section 2.2 discusses some prior work in the area of mechanism and policy separation.

Gateways between heterogeneous networks extend the notion of Internet routers to encompass the embedded domain. In doing so, the differences between enterprise and embedded networks must be taken into account. Enterprise networks generally have highly variable latency, high bandwidth, and the ability to send relatively large messages. Embedded networks generally have low bandwidth, smaller messages, and the ability to enforce real-time deadlines for message delivery. These characteristics, with examples of each network type, are described in greater detail in Section 2.3.

The notions of time-triggered and event-triggered architectures are closely linked to the notions of enterprise and embedded networks, respectively. The differences in these archi-

tectures is described in Section 2.4. In this work, gateway applications are examined using time-triggered data flowing from an enterprise network to an embedded network.

A great deal of effort has gone into the design of high performance routers and routing protocols for managing traffic in enterprise networks. Policies for queue mechanisms are identified from prior work in the area of queue management. The sources of these existing policies are described in Section 2.5.

In order to evaluate the performance of gateway networks, we must have an application to study, and we must have a way to observe that application in action. Discrete event simulation is employed to model systems with gateways. Section 2.6 gives some background on the use of discrete event simulation. The particular system studied is a traffic control system from the field of traffic flow modeling, an area of active research with a body of work on microsimulation (where the behavior of each vehicle is simulated separately). The microsimulation approach is well-suited for studying gateway mechanisms because each vehicle is modeled, so individual vehicle models receive network traffic which is passed through a gateway. This topic of traffic flow modeling is described in more detail in Section 2.7.

Chapter 3

Simulating Gateways with Abstract Network Models

In this chapter, we describe the simulation framework used to evaluate various mechanisms with abstract network models, the apparatus used to collect input data for the simulations, and the metrics recorded to evaluate and compare different mechanisms. Abstract network models are useful because they provide insight about the operation of gateway systems and the problems that can arise without having to model specific protocols.

3.1 Simulation Framework

In order to evaluate the performance of the various queue management mechanisms, we have developed a discrete event simulator in Java. The simulator executes scheduled events with microsecond granularity. Simultaneous events (events that occur at the same simulation time) are executed in pseudo-random order. If two sequences of events with the same period are started at the same time, the events will be processed in different (random) orders in each period. Without the pseudo-random ordering, the startup order of different processes would create an implicit priority in their execution.

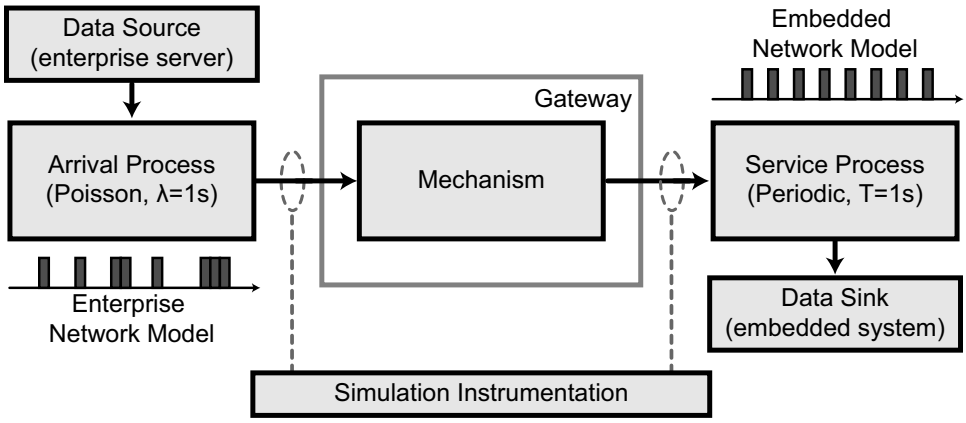


Figure 3.1. Event simulator with abstract network models

3.2 System Models

In order to model gateway systems, the simulator uses abstract network models to represent enterprise and embedded networks. The transmitting network is modeled as an arrival process. The receiving network is modeled as a service process. The type of process chosen for the arrival and service processes reflects the type of networks being modeled. An example simulator configuration is shown in Figure 3.1. The embedded network is modeled with a periodic process that reflects the real-time characteristics of the embedded network. The enterprise network is modeled with a Poisson process. The Poisson process is chosen to capture the non-real-time characteristics of enterprise networks. Although these models are simple, they still provide useful insight into the behavior of gateway mechanisms. The mechanism in the gateway could be a queue mechanism, a filter mechanism, or any other mechanism class being studied. Only one mechanism is used in any given simulation run. Because the simulation is implemented in Java, any mechanism that can be expressed in code can be implemented. This capability allows the reuse of existing code (for example, algorithms for interpolation and extrapolation).

The simulation models an arrival process that delivers data to the queue and a service process that removes messages from the queue. Each process can be specified to be deterministic (e.g. periodic) or to occur randomly according to a probability distribution.

All the random elements or sequences in the simulation are generated using the deterministic pseudo-random number generator provided by the *java.util.random* package [35]. The software can repeatably generate the same pseudo-random sequence from a given seed value.[1] Thus, the same pseudo-random arrival sequence can be recreated and applied to gateways with different mechanisms to allow for a fair comparison of their performance.

3.3 Input Data Collection

In these experiments, we are concerned with state-oriented time-triggered data streams. We have developed a data collection system that uses the automotive standard OBD-II diagnostic interface [36] to record the speed of a vehicle during operation. The system was used to collect four different data sets to use as inputs to the simulation. The data collection system provides sets of input data values that are used in the abstract network simulations. The collected data is resampled to periodic intervals required by the abstract network simulation. Thus, the timing of the message arrivals in the simulation is determined by the abstract network model, not by any timing information collected while driving.

Each data set represents a different driving scenario. These scenarios provide variety in the character of the data that is used as input to the simulation. The data sets are described and plotted below.

[1] The repeatability is derived from a requirement of the Java API Specification to use a specific algorithm. It is possible that a particular Java VM implementation fails to meet the API contract. The results in this work were executed on the Sun/Oracle Java VM, which was tested to meet the repeatability requirement.

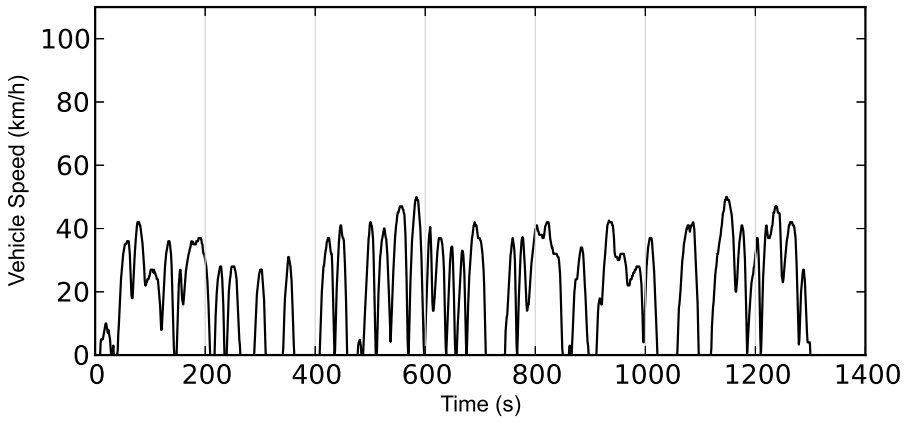


Figure 3.2. Squirrel Hill Data Set

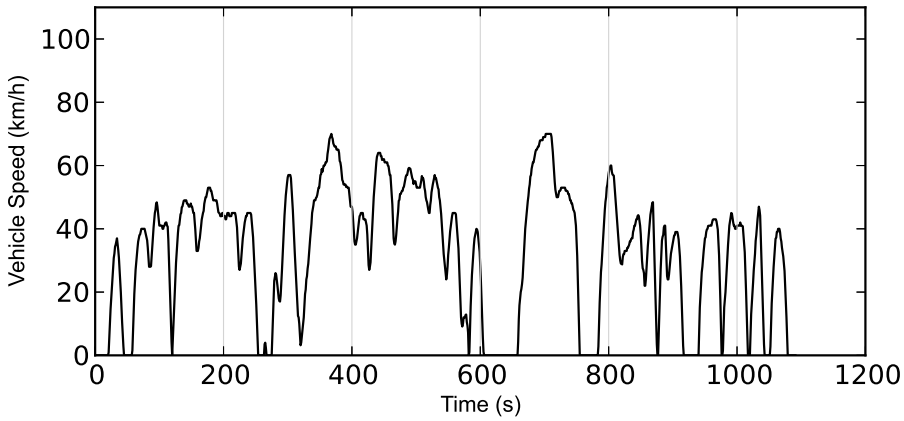


Figure 3.3. Beechwood Data Set

The first data set is shown in Figure 3.2. The data is obtained from a neighborhood driving scenario with low speed limits and many stop signs and stop lights. The frequent starts and stops are visible in the graphed data.

The second set (shown in Figure 3.3) is obtained from neighborhood roads that are not highways, but have fewer stop signs and lights and higher speed limits than the first data set.

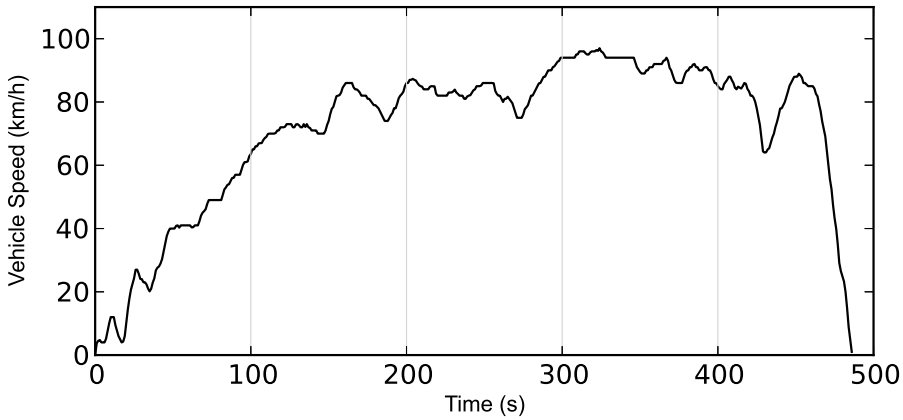


Figure 3.4. Monroeville I Data Set

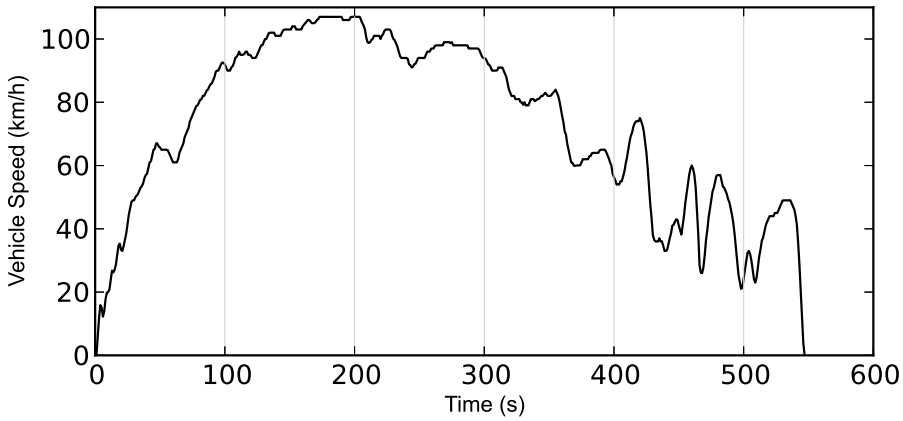


Figure 3.5. Monroeville II Data Set

The final two sets are obtained from highway driving scenarios. The Monroeville I Data Set is taken in relatively light traffic (Figure 3.4). The Monroeville II Data Set (Figure 3.5) is taken in heavier traffic. The effects of highway congestion can be seen in the reduced speed and increased variation starting around 400 seconds.

These data sets are useful for mechanism evaluation because they represent time-varying data from actual road scenarios. An application dealing with road traffic would be sending

and receiving similar data, for example, by reporting the average speed of traffic in the upcoming road segment. All four data sets are used to evaluate the various mechanisms.

3.4 Metrics

There are several metrics which are recorded during the simulations. Each metric is aggregated over all values for a single trial, so it produces a single value for each trial. In the results presented in Chapter 6, we compare the distribution of these metrics for experiments using different mechanisms.

- *Maximum queue length* is the maximum queue length observed during a single trial.
- *Average queue delay* is the delay for each message (time between arrival at the queue and departure from the queue) averaged for all messages in a single trial. Dropped messages are not factored into this metric, and neither are duplicate deliveries that occur because of the mailbox policy (see Section 4.1.2).
- *Dropped message count* is the total number of dropped messages in a single trial.
- *Mean squared error* is a metric designed to capture how well a gateway mechanism preserves the data sequence. It is computed by recording the point-by-point difference between the original data sequence and the data sequence output by the gateway. The average of the square of these errors is computed over the whole run.

The first three metrics pertain specifically to queues. They are used to study and compare queue mechanism performance. However, these metrics are based on the structure and function of queue mechanisms, and they are not well-defined for a filter mechanism which can arbitrarily modify or generate outputs. The final metric, mean squared error (MSE)

requires only the input data and the output sequence and does not make assumptions about the structure of the mechanism itself. Thus, it can be used to compare the performance of different types of mechanisms.

3.5 Experimental Setup

The abstract network experiments presented in this work are designed to model the enterprise-to-embedded scenario described in Section 1.2.1. Internet traffic has been shown to be bursty [37]. [38] discusses several delay models that can be used to model an enterprise network: constant delays, independent random delays (e.g. the Poisson process), and Markov chain models (which capture the effect of network load on the distribution of delays). We model the arriving data with a Poisson random process because it does not require an exploration of the additional parameters of the Markov chain models, but it captures the non-real-time nature of the arriving data.

For each experiment, the simulator is configured with a Poisson arrival process with mean inter-arrival time of one message per second and a periodic service process also with a period of one second. The choice of one second is arbitrary, but the important feature of the experiment is that the arrival and service processes have equal mean rates. Each experiment employs one of the vehicle speed data sets described in Section 3.3 as input data. The simulated gateway is configured with a particular mechanism (e.g. finite queue of length 50 using the Drop Oldest overflow policy). A trial is a single run of the simulation and produces a single value for each of the metrics described in Section 3.4. In each trial, a different pseudo-random arrival sequence is applied to the gateway. The sequence of data *values* delivered to the gateway in each trial is the same (such as those pictured in Figure 3.3). Only the *timing* of the arrivals changes from trial to trial. Each experiment consists of 5,000 trials. Bootstrap analysis of the results shows that percentiles of each metric have

a maximum 95% confidence interval of 2%. An experiment set is a series of experiments performed with different mechanisms (e.g. a set of experiments on queues of length 10, 20, and 50) intended to compare the performance of the mechanisms with respect to one or more metrics. Experiment sets are repeated using each of the four data sets as inputs.

3.6 Experimental Results

Each experiment yields a set of 5,000 values for each metric. These values can be summarized by a boxplot, as shown in Figure 6.1. The experiment set yields a summary boxplot for each mechanism. Comparing the boxplots allows the relative performance of each mechanism in the experiment set to be evaluated. The results of the experiments described here are presented in Chapter 6 and Appendix A.

3.7 Summary

This chapter describes an approach for studying gateway mechanisms in enterprise-to-embedded scenarios using abstract network models. In this context, the arrival process (a Poisson random process) is a model of the enterprise network, and the service process (a periodic process) is a model of the embedded network. The gateway mechanism receives data during an event on the arrival process and emits data during an event on the service process.

We construct discrete event simulation experiments using these abstract models (see Section 3.1 and Section 3.2). Vehicle speed data is collected from actual driving scenarios (see Section 3.3) as inputs to discrete event simulations. The performance of various mechanisms is studied by measuring the mean squared error between the original data sequence and the output of the gateway during the simulations (see Section 3.4). A distribution for

the MSE is collected over multiple simulation runs with different random arrival sequences (see Section 3.5 and 3.6).

These distributions provide insights into the performance of various gateway mechanisms and policies, although this chapter only describes the way abstract network simulation experiments are constructed. Chapter 4 describes the mechanisms and policies to be evaluated, while Chapter 6 gives evaluation results for abstract network experiments and a discussion of our insights into gateway mechanism selection.

Chapter 4

Gateway Mechanisms

Gateway mechanisms are implementations of algorithms that are used to manage the flow of data through the gateway. Here we focus on gateways in the enterprise-to-embedded scenario with time-triggered, periodic data. In this kind of system, the enterprise network introduces delays in a periodic data stream. The fundamental task of the gateway mechanism is to receive the perturbed data from the enterprise network and recover a timing and data sequence that is as close to the original (unperturbed data) as possible, as shown in Figure 4.1. Other gateway scenarios are described in Section 1.2.1, and descriptions of enterprise and embedded networks are given in Section 2.3.1 and Section 2.3.2.

Usually, the gateway mechanism will be implemented in software on a general-purpose computing platform, although FPGA or custom chip implementations are also feasible as the technology and understanding of gateway mechanisms matures. Our focus here is not on implementation of a physical device, but on understanding the ability of different mechanisms to handle data and the impact of mechanism and policy choice on application performance.

This chapter describes several mechanisms and the policies that can be chosen by a designer to obtain improved performance of the gateway. Queue mechanisms are studied since they are a well-known mechanism that is currently being used in almost all similar applications (e.g. routing). Filter mechanisms are a new mechanism we have developed

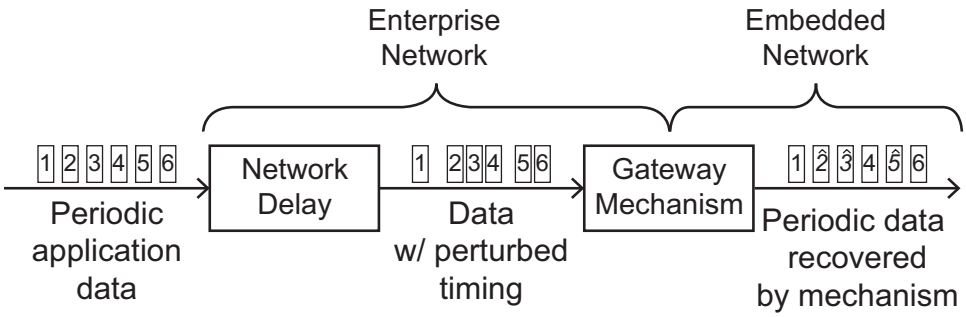


Figure 4.1. Visualization of gateway data flow for enterprise-to-embedded gateway scenario with time-triggered, real-time data.

to address specific shortcomings of queues in the enterprise-to-embedded scenario. Other mechanisms and policies are certainly feasible, but the performance of each mechanism and policy will be dependent on the characteristics of the application. Given the wide variety of application requirements and application characteristics, rather than attempting to provide an exhaustive list of mechanisms and policies, we focus on a few mechanisms and policies that are suited to the traffic applications we study in Chapters 3 and 7. This research provides a workflow that shows how mechanisms and policies can be evaluated for a specific application.

In this discussion of mechanisms, we make the distinction between generic mechanisms and application-aware mechanisms. Queues are generic mechanisms. They are also the standard mechanism used in most routing applications. They act on packets without knowing the contents of the packet or the purpose of the packet. An exception is Deep Packet Inspection routers, which do analyze packet content, although the focus of these routers is on identifying the higher-level protocol (e.g. FTP, BitTorrent) for security purposes and for prioritizing traffic. The filter mechanism discussed here has a stronger notion of application awareness than even Deep Packet Inspection. We assume that the gateway designer has *a priori* knowledge of the applications using the gateway and the format and seman-

tic meaning of the information contained in the packets. For example, if the gateway is used for a speed control application (as in the case study in Chapter 7), then the gateway designer knows *at least*: (a) the format of the packets[1], (b) that the data in the packets represents a speed value, and (c) the units of the measurement used. The designer also should have information about the expected characteristics of the networks, which is useful for interpreting the results of the Independent Delay Analysis (see Chapter 5).

Section 4.1 describes queue mechanisms and the policies for limiting queue length and determining how to handle overflow and underflow situations. Section 4.2 describes the filter mechanism (which we have developed based on insights from queue mechanisms) and describes the policies for specifying estimation models in filter mechanisms.

4.1 Queue Mechanisms

Since queue mechanisms are applied successfully in Internet routers, they are a logical starting point for gateway mechanisms. All the queues discussed in this paper use the first-in-first-out (FIFO) queue discipline. Data arriving from the transmitting network is stored in arrival order. When the time comes to send data on the receiving network, the oldest value is sent and then removed from the stored data.

There are three policies related to queue mechanisms:

- **Length policy:** the maximum number of data items to be stored
- **Underflow policy:** how to handle the data items when the queue is empty
- **Overflow policy:** how to handle the data items when the queue has reached the maximum length

[1]The format of the guidance packets used in the case study is described in 7.3.2.2.

4.1.1 Queue Length Policies

For queue length policy, we consider both **bounded queues**, which are constrained to a certain maximum length and fixed for the duration of an experiment, and **unbounded queues**, which are allowed to grow to any length. While it is not possible to implement a truly unbounded queue in practice, we assume for the sake of analysis that the storage capacity of any practical implementation can be made arbitrarily large. For the data rates and arrival processes used in our experiments, we observe that the behavior of longer bounded queues begins to converge to the unbounded queue around length 50. For queue management policies, we consider behavior during underflow and overflow situations.

4.1.2 Queue Underflow Policies

Queue underflow policies describe how the queue mechanism handles a service event when there is no data in the queue. In an event-triggered architecture, having an empty queue means simply waiting for incoming messages to arrive. However, the embedded network assumes a time-triggered architecture, which creates periodic service events that may exhaust the queue, especially if the data from the enterprise network is not periodic (e.g. a Poisson random process). A need for an underflow policy could also arise in the embedded-to-embedded context due to jitter in network schedules, although study of the embedded-to-embedded scenario is beyond the scope of this work.

The first underflow policy is a **mailbox policy**. This term is used because of the similarity to the mailbox implementation used in CAN controllers [26]. The mailbox is separate from the queue's data storage and is able to store a single data element (i.e. queue entry). The mailbox holds a copy of the value from the most recent service event. If no new

value is available from the queue, the mailbox value is sent instead. Depending on the implementation details, the mailbox value may be marked with a staleness indicator.

The second policy is an **invalid value policy**. Whenever the queue is empty, a special value is sent, or a flag is set in the message that lets the receivers know it is invalid.

The third policy is a **send no value policy**. Under this policy, no message is sent on the receiving network. This policy may not be feasible for some embedded networks. For example, both TTP and FlexRAY (static segment) have fixed TDMA schedules that require messages to be sent periodically. Other networks may require transmissions of a certain frequency to maintain edge synchronization.

All the experiments described in this work use the mailbox policy as their underflow policy. For time-triggered receiving networks studied here, if an invalid value is sent or no value is sent, the application is going to continue to use the last valid value (e.g. the set point for an actuator remains at the last received value), so the net effect of these other policies is similar to that of the mailbox policy. However, in an event-triggered receiving network (such as in the embedded-to-enterprise scenario), the mailbox policy could result in the system interpreting repeated values as additional events. In this case, the null or invalid message policies might be preferred.

4.1.3 Queue Overflow Policies

Queue overflow policies describe the action to be taken when the queue exceeds its designed maximum length. These policies only apply to bounded queues. For an unbounded queue, an overflow condition cannot occur since we assume arbitrarily large system resources. Some policies described here may drop more than one message or cause the *incoming* message to be dropped. When a message is dropped by the gateway, the fact that the message was dropped is not reported to either the sending or receiving network.

For example, if messages on the transmitting network are using TCP, the gateway would send the TCP ACK on the transmitting network, then would discard the message without sending it on the receiving network.

We have identified four queue overflow policies which are described below:

1. The **Drop Newest Policy** requires that the newest message (the arriving message) be dropped. This policy is similar to the active queue management technique known as Drop Tail [13], which has been used in Internet routers.

2. The **Drop Oldest Policy** requires that the oldest message (i.e. the message at the head of the queue) be dropped. This policy is more useful for state-oriented messages where the more recent messages contain a more accurate description of the current system state and is similar to the Drop Front congestion control technique proposed in [39].

3. The **Drop Random Policy** requires that a random message be dropped from the queue when an incoming message arrives at a full queue. The incoming message is included in the pool of candidate messages to be dropped. This policy is similar to the Random Early Drop technique [40].

4. The **Drop All Policy** requires that the queue be flushed (completely emptied) when a new message arrives at a full queue. The arriving message is not dropped, but all the messages already stored in the queue are dropped.

While the Drop All policy is novel, the remaining three are an application of existing queue management techniques to embedded system gateway queue mechanisms. However, as noted, router queue management would *not* send a TCP ACK for a dropped message (thus causing the sender to reduce its sending rate in accordance with TCP protocol). The

gateway mechanism applies the same policies, but with the goal of managing the data in gateway, not link congestion.

4.1.4 Insights on Queue Underflow

A full evaluation of the performance of queue mechanisms is presented in Chapter 6. However, there is a key insight into the behavior of queue mechanisms that we present here to give context for the filter mechanisms discussed in the next section. An obvious and well-understood property of queues is that the longer the queue is, the longer the delay it imposes on the packets that pass through it. However, the abstract network simulation results (see Section 3.5 for details) reveal an important property of queue mechanisms. When there is a long delay, the queue must provide data to the periodic service process regardless of the fact that the queue has been emptied, hence the underflow policies discussed above. The delay is usually followed by a burst of data, including the delayed data. However, the periodic service process continues to extract data at the same rate. The late data goes into periodic timeslots that should have been occupied by later (not delayed) data. The burst increases the length of the queue and delays all the subsequent arriving data until a gap in the transmitter network data allows the queue to shorten.

This process is illustrated in Figure 4.2. Part (a) shows the ideal case where periodic inputs and outputs are the same. In this case, there should be no steady-state accumulation of messages in the queue. However, the arrival process is not periodic. Messages that were transmitted periodically become clumped together as pictured in part (b) of the figure. When the first three messages arrive in a burst, they are queued and delivered in their appropriate time slots. Because of the long quiescent period between the bursts, the fourth message has not arrived when the fourth time slot comes up at the output, so the third message is sent again (per the mailbox policy). When the fourth message does arrive, it is

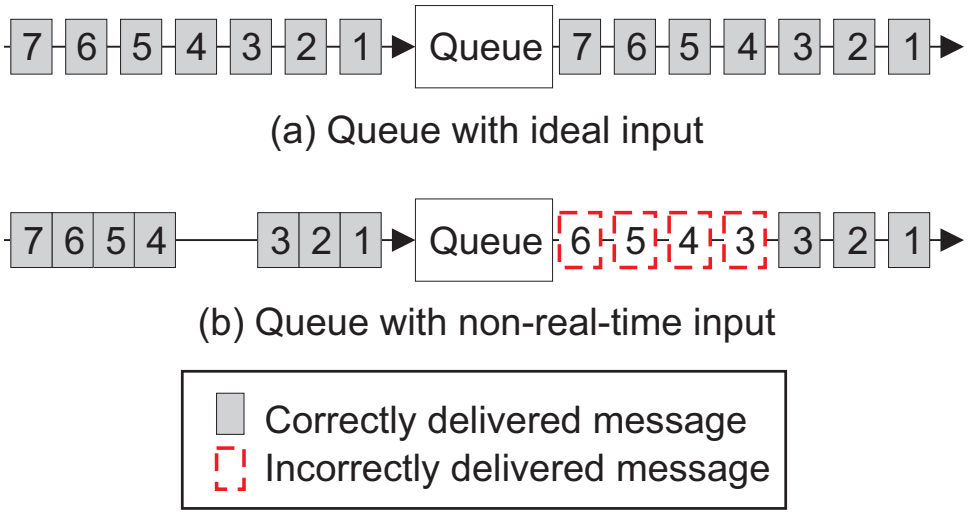


Figure 4.2. Illustration of Queue Underflow Due To Non-real-time Inputs

sent in the time slot where the fifth message should go, and the fifth message is delivered in the sixth slot, and so on. The time slot missed by the fourth message cannot be recovered, and the steady state size of the queue has increased by one, adding one message period delay to each delivered message. This process can happen repeatedly, causing the queue to grow longer and longer as the result of either normal timing variations or a malicious attacker purposely clumping message arrival times.

This insight leads directly to the design of the filter mechanism which is discussed in the next section.

4.2 Filter Mechanisms

A filter mechanism is a mechanism that we have developed to address the problem of queue underflow in the queue mechanism. Similar to the queue mechanism, the filter stores arriving packets in a FIFO data structure and delivers them to the service process when requested. The key difference in the filter behavior comes when the service process requests a packet and the filter's FIFO is empty. The filter responds to the service event in accordance

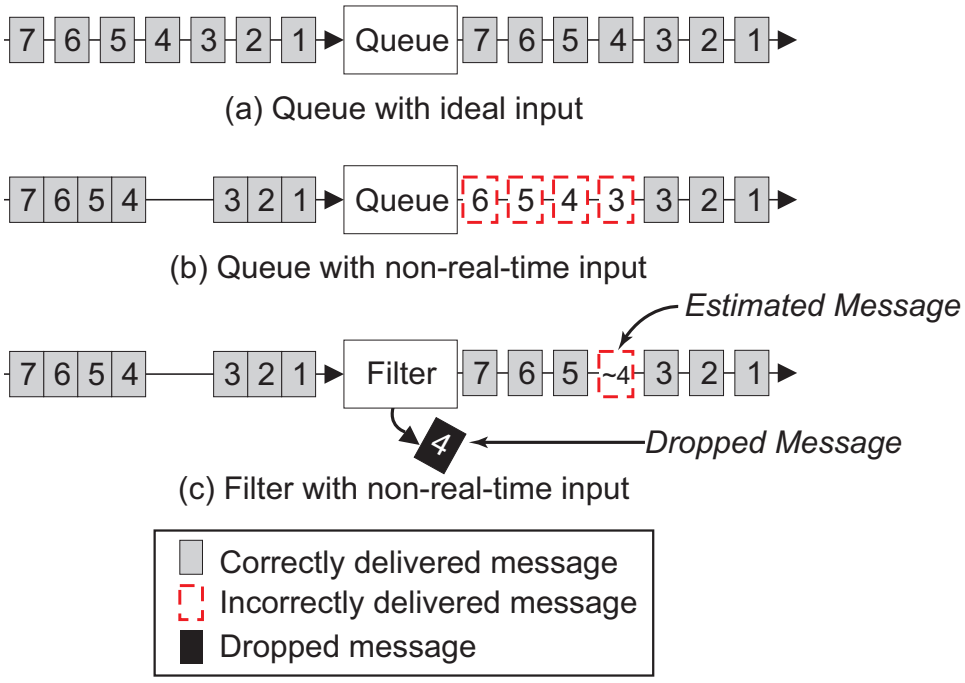


Figure 4.3. Demonstration of a Filter Mechanism Mitigating Queue Underflow

with its underflow policy, while keeping track of the number of messages that have been missed. As delayed packets arrive, they are discarded instead of being stored in the FIFO for eventual delivery to the service process, preventing the FIFO from building up length due to underflow, as would happen with the queue mechanism.

Figure 4.3 shows an example of the filter mitigating the queue underflow problem. Parts (a) and (b) are the same as Figure 4.2 discussed in the previous section and show the queue underflow causing delay. Part (c) shows how the filter handles this problem. When underflow occurs and the fourth message is not available in the fourth time slot, the filter mechanism produces an estimate of the missing data. When the fourth packet arrives, it is *not* delivered in the fifth time slot. In fact, it is discarded and never delivered to the service process.

Filters differ from queue mechanisms in one other important way. The underflow mechanisms also use incoming message data to provide estimate values delivered when the filter FIFO is empty. This estimation technique is what makes the filter an application-aware mechanism.

The FIFO structure in filter mechanisms is treated as unbounded, so neither a length policy nor an overflow policy is needed.

4.2.1 Filter Underflow Policies

Defining a filter underflow policy consists of selecting the data model used for estimation and defining any parameters required by the model. Although delayed messages that are dropped from the FIFO are never sent on the receiving network, the data may still be incorporated in the data model to improve the estimates of later data points.

The models presented here are not an exhaustive list of all models that can be used for estimation in filters. Many other models are possible based on regression and curve fitting, probabilistic models, state estimation techniques (e.g. Kalman filters), and many other approaches. Some models, such as the linear regression model, were evaluated and discarded for our case study application based on the results of the Independent Delay Analysis (see Figure 5.1 and the discussion in Section 5.2. However, for other applications, these and other models should be considered based on the characteristics of the system, networks, and data involved.

The underflow policy model is defined as a function of the current time delay and past values received by the gateway, expressed as:

$$\hat{s_d} = A(d, s_1, \ldots, s_i) \tag{4.1}$$

where $\hat{s}_{d,i}$ denotes the model estimate based on the delay d and the history of values s_1 to s_i. The delay d is the difference between the timestamp of s_i and the time at which the next service event will occur. The length of the history required by a policy model varies depending on the type of model. There is no requirement that models keep more history than needed for their estimates. For example, a linear extrapolation model might only retain the values $s_i - 1, s_i$. As with other mechanisms, we assume that a gateway implementation will have sufficient resources to store and process the required amount of history information for any policy models used. The history is updated every time a new packet arrives at the gateway regardless of whether the arriving value is delivered or dropped.

The remainder of this section describes several data model policies that can be used in filter mechanisms.

4.2.1.1 Extrapolating Data Models

Extrapolation techniques have the benefit that they rely on numeric data only and do not need additional parameters or process modeling, so they are simple to apply with little or no understanding of the system that generates the data. We assume that a timestamp accompanies the data so that the timestamp can be used along with the data to obtain curve fits. Using timestamps generated at the data source is preferred over using the arrival time at the gateway, which is likely to be affected by network latency.

The simplest type of extrapolation policy is the **Constant Extrapolation Policy**. This involves simply repeating the last value that was received. It is given by the equation:

$$\hat{s}_d = CE(d, s_i) = s_i \tag{4.2}$$

The **Linear Extrapolation Policy** is one such filter policy. It has the benefit of maintaining the upward or downward trend of the data, and it can be useful when network delays are small. It is given by the equation:

$$\begin{aligned} \hat{s}_d &= LE(d, s_1, s_2) \\ &= s_1 + (s_1 - s_2)\frac{d}{T} \end{aligned} \tag{4.3}$$

For efficient computation, we use an implementation of Neville's algorithm from [41] to obtain quadratic and cubic fits to data values. Polynomial fits tend to be better within the data, so the higher-order models tend to be poor estimators. Thus, the **Quadratic Extrapolating Model Policy** and **Cubic Extrapolating Model Policy** are of little use for the applications studied here, although it is possible that they might be useful in other systems.

4.2.1.2 Hybrid Data Models

Experiments with extrapolating models lead to some insights about when different models are effective, which leads in turn to two hybrid data models. These hybrid models are still numeric in nature and are based on empirical observations, not system models. They do provide additional tuning parameters whose value must be chosen based on analysis of the system behavior.

The first model policy is the **Piecewise Combined Policy**. This model is based on the observation that a linear extrapolation estimate can be better than a constant estimate when delays are small, but that constant estimates are better for long delays. The model switches between a linear extrapolation estimate and a constant extrapolation estimate when delay

exceeds a fixed threshold. It is given by the equation:

$$\begin{aligned}\hat{s}_d &= PC(d, s_1, s_2) \\ &= \begin{cases} LE(d, s_1, s_2) & \text{for } d < d_{max} \\ CE(s_1) & \text{otherwise} \end{cases}\end{aligned} \tag{4.4}$$

Note that the parameter d_{max} is fixed for any particular implementation, but may be varied by the system designer.

The second hybrid policy is the **Decaying Linear Extrapolation (DLE) Policy**. This model attempts to refine some shortcomings in the Constant-Linear Extrapolation model, namely that the switchover between the linear and constant estimates can result in a discontinuity in the output values. It is given by the equation:

$$\begin{aligned}\hat{s}_d &= DLE(d, s_1, s_2) \\ &= \begin{cases} s_2 + (s_2 - s_1) \cdot \sum_{i=1}^{\frac{d}{T_s}} \frac{d_{max} - (i \cdot T_s)}{d_{max}} & \text{for } d \leq d_{max} \\ \hat{s}_{d_{max}} & \text{for } d > d_{max} \end{cases}\end{aligned} \tag{4.5}$$

This model is developed using the Independent Delay Analysis method. The analysis that led to its creation is described more fully in Section 5.3.

4.3 Summary

This chapter identifies mechanisms and policies that can be used to manage message data in the gateway. There are two mechanisms discussed here: queues and filters. Queues are mechanisms that are already being used to manage information flow in Internet routers.

They are generic mechanisms, i.e. they (for the most part) treat packets only based on source and destination information, and not based on the semantic content of the messages.

In Section 4.1, we identify three policies that can be used to configure queues: length policy, underflow policy, and overflow policy. Queue length policy sets a limit on how long the queue is allowed to grow. Queue overflow policy determines which message is dropped when the maximum length is reached. Queue underflow policy determines how the mechanism behaves when the embedded network requests the next message while the queue is empty.

Simulating queue mechanisms with abstract networks has led to a particularly important insight regarding queue mechanisms: when queues underflow, a periodic timeslot is missed on the embedded system, but the late message is still transmitted in a later slot. The delayed message increases the length of the queue and the delay for each succeeding message. Section 4.1.4 describes the underflow-delay phenomenon in more detail.

Filters are a mechanism that we have designed specifically to mitigate the underflow problem with queues. The main idea of a filter mechanism is that estimated values are used to replace delayed messages, and the delayed messages are discarded, eliminating the underflow delay problem. Filters are necessarily application-aware mechanisms. They must be aware of the periodic nature of the message flow in order to drop delayed messages. To estimate values for delayed messages, they must also have semantic information about message content.

Filters are configured by their underflow policies, which consist of a data model used to estimate delayed messages. Several model policies are described in Section 4.2, including the decaying linear extrapolation policy, which was developed using the Independent Delay Analysis method in Chapter 5.

The queues and filter mechanisms (and their policies), as described in this chapter, are evaluated using abstract network models in Chapter 6 and the traffic control case study in Chapter 7.

Chapter 5

Independent Delay Analysis

In Section 4.2.1, we described the underflow policy for filter mechanisms. The underflow policy includes a data model that is used for estimating values output by the filter mechanism. This chapter describes Independent Delay Analysis, a technique we have developed for comparing these data models.

Since data models for filter underflow policies can be based on extrapolation, regression, moving averages, machine learning, or any other estimation technique, we need a way to understand and compare them that is (a) not dependent on the structure of the model and (b) easy to carry out for a large number of models. Our technique focuses on understanding the effect of delay on the performance of these models. It is a lightweight numerical analysis technique that can be performed using tools such as Matlab or Mathematica or the SciPy python libraries. Since many of these estimation techniques are already implemented in these tools, applying them as data models can be relatively easy.

The result of the analysis shows the performance of each data model for a range of fixed delays, allowing the selection of the best data model depending on the delays that are likely to be observed in the application. Although the results do depend on application-specific data, they are independent of the application network model (hence the name independent delay analysis). Thus, if an application is in the design phase, the results could be used to

guide the selection of networks by providing insight into the delay performance of various data models.

5.1 Method Description

This technique, Independent Delay Analysis (IDA), explores the effect of delay on data model performance in a way that allows selection of the best data model from among a set of known candidates. The key objective for this approach is to measure and compare the error performance of data models using different, fixed delays. The results can be used to select the appropriate filter underflow policy for a particular network, based on the delay characteristics of that network.

The steps in the IDA method are:

1. Choose a representative data set.
2. Choose a data model.
3. Use the data model to build a series of estimates based on fixed delays.
4. Compute the error for each estimate.
5. Repeat steps 1-4 with various models, then compare their error performance to select the best model or model(s).

We now explain each step in greater detail.

1. *Choose a representative data set for the application.* The input data set, S, of length, n, is denoted by $\{S : s_1, \ldots, s_n\}$. Since the values in the data set are sent periodically, the index i on a value s_i is essentially a timestamp, which keeps the notation straightforward. It would be relatively simple to extend this method for non-periodic data by defining the data sample s_i as a tuple with an explicit timestamp.

2. *Choose a data model.* The model is defined as a function $F(l, s_j, \ldots, s_k)$ which produces $\hat{s}_l$, an estimate of the application data at time l. The data model policies used in this research are described in Section 4.2.1.

3. *Use the data model to build a series of estimates based on fixed delays.* The model function is used to construct a series of estimated values $\hat{S}_\delta : \hat{s}_{\delta,w+1}, \ldots, \hat{s}_{\delta,n}$ where δ is the fixed delay and $\hat{s}_{\delta,i}$ is given by Equation 5.1:

$$\hat{s}_{\delta,i} = F(\delta, s_1, s_{i-\delta}) \tag{5.1}$$

   That is, each estimated value $\hat{s}_{\delta,i}$ is the estimate that would be given by the model at time i if the model were receiving all the input data from S with a delay of δ.

   Although the definition of $\hat{s}_{\delta,i}$ in Equation 5.1 includes the entire (delayed) history of S, there are a number of ways a practical implementation could avoid the need for an infinite history buffer. For example, some models will only use a subset of the most recent points. A linear extrapolating model needs only the two most recent points for a linear estimate. An equivalent, stateful implementation of the model that is updated with each subsequent value of S could also be implemented to avoid the infinite history buffer. These practical considerations do not hinder the usefulness of Equation 5.1 as a way to think about filter model policies.

   $\hat{S}_\delta$ begins at time $w+1$, where w is the warm-up time that ensures that the model has enough data to begin making estimates. For example, a constant-order extrapolating model needs two points for an estimate, so w would be two in that case.

4. *Compute the error for each series of estimates.*

The mean squared error, e_δ, is computed between the input sequence S and each estimated sequence $\hat{S}_\delta$:

$$e_\delta = \frac{1}{n-w} \sum_{i=w+1}^{n} \left(s_i - \hat{s}_{\delta,i}\right)^2 \tag{5.2}$$

The error e_δ is computed for a range of values of δ, giving $E_F : e_1, \ldots, e_{\delta_{max}}$, a complete set of error metrics for the model F.

5. *Compare performance among models.* Steps one through four are repeated with each model under consideration, then the respective E sequences are compared to identify the model with the lowest e_δ for each value of δ.

5.2 Interpreting IDA results

Figure 5.1 shows the results obtained by applying the IDA method for several different models using the Beechwood vehicle speed data set (see Figure 3.3) for delays from 1 to 20 seconds. The Root Mean Squared Error (RMSE), the square-root of MSE, is plotted (instead of MSE) for more convenient scaling. Each line in the figure represents a plot of E_A for a particular model A. We have included the four extrapolation models described in the previous section, as well as a linear regression model which makes a linear estimate based on a least-mean-squared fit of the five most recent data points. To read the results, we look for the curve with the lowest error. Among these models, the linear extrapolation model has the best error performance for delays up to seven seconds, but for eight or more seconds, the constant extrapolation model is best.

Approximately 74% of the non-zero delays in the simulation are above eight seconds. So during the simulation, the system spends most of its time in the region on the right side of

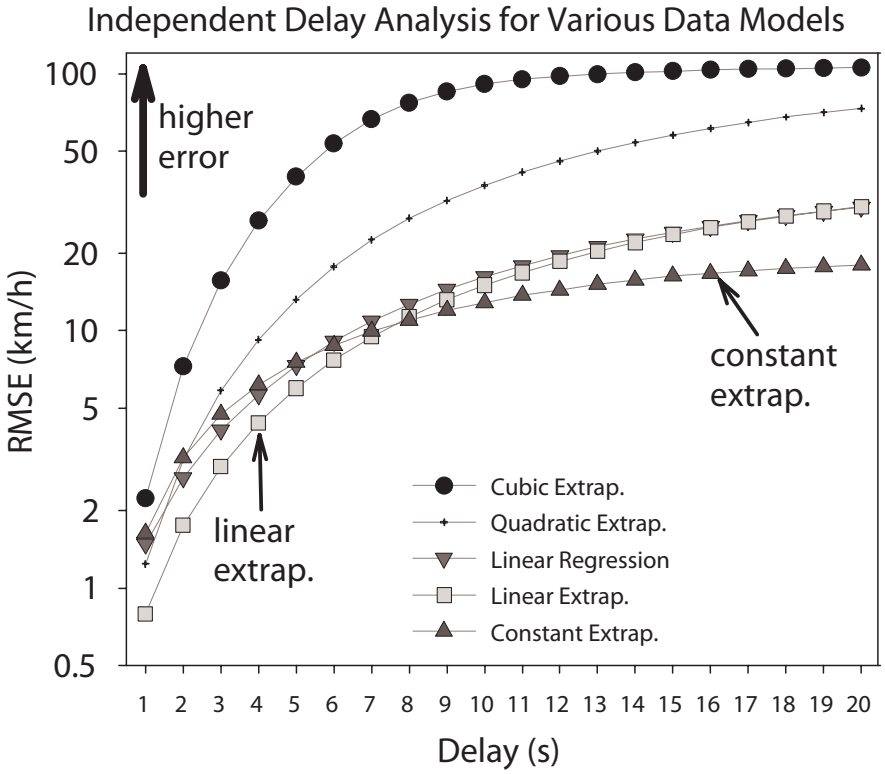


Figure 5.1. Results of the Independent Delay Analysis for various data models. This graph shows the error performance of each model for different fixed delays.

the IDA graph, where the constant-order model is best. Furthermore, longer delays result in a higher error, so they contribute more to the overall error than the lower delay values.

In addition to seeing which models are best, one can also see which models are similar. For example, the linear extrapolation and linear regression models have similar performance. In an embedded context (where systems are often constrained in terms of computation or memory requirements), if two models have similar performance, but one model has a smaller computation or memory requirement, then the simpler model might be preferred, even if the more complex model has slightly better MSE performance.

Because the IDA results are based on fixed delay computations, the results are independent of any network properties of a particular gateway configuration. The shape of the graph in Figure 5.1 (including the crossover point between seven and eight seconds) is only a function of the data models and the input data.

Network independence is useful because the IDA results show how network delay properties will affect model behavior. For example, if the linear model is being used, then changes to the system that result in many delays longer than five seconds should be avoided. However, if the cubic extrapolating model is being used, there is little difference between a system with delays around 12 seconds and one around 20 seconds.

5.3 Using IDA to Combine Data Model Policies

The IDA results in Figure 5.1 show how delay affects the performance of different data models. For some applications, there may be one model that is always better than the others. If that is *not* the case, then a combined model can be constructed [42].

A model with the lowest error for each delay value is desired. The simplest combined model is one whose model function is mathematically defined as the piecewise combination of the model functions of the best models for each range. For example, based on the delay values from the IDA results in Figure 5.1, the piecewise combined model should be:

$$\hat{s}_d = \begin{cases} LE(d, s_1, \ldots, s_i) & \text{for } 1 \leq d \leq 7 \\ CE(d, s_1, \ldots, s_i) & \text{for } 7 < d \end{cases} \tag{5.3}$$

where CE and LE are the model functions for the constant and linear extrapolation models, respectively.

Since the model is defined as a piecewise combination of two already-tested models, repeating the IDA analysis is not necessary. It will exactly track the linear model for $d \leq 7$ and exactly track the constant model for $d > 7$.

One problem with the piecewise model is that it will result in a discontinuity when switching from one model to the other. Although such an approach might be appropriate for some applications, it is at odds with the inertia-backed physical processes that produce these data. A smooth model that acts like a linear extrapolation model for short delays and a constant extrapolation model for long delays is preferred, so we propose a combined data model policy which is called the Decaying Linear Extrapolation (DLE) model. The model produces a linear estimate which decays to a constant value after a maximum delay value, d_{max} is reached. This model uses a concept of a forgetting factor in the area of adaptive filtering [43]. The model function of the DLE model is given by:

$$\begin{aligned} \hat{s}_d &= DLE(d, s_1, s_2) \\ &= \begin{cases} s_1 + (s_1 - s_2) \cdot \sum_{i=1}^{\frac{d}{T}} \frac{d_{max} - (i \cdot T_s)}{d_{max}} & \text{for } d \leq d_{max} \\ \hat{s}_{d_{max}} & \text{for } d > d_{max} \end{cases} \end{aligned} \tag{5.4}$$

The underlying idea of the model function is this: the slope between the last two points $(s_1 - s_2)$ is attenuated by a factor that increases linearly for each additional period of delay for delays up to d_{max}. Above d_{max}, the model estimate no longer changes (until new data arrives). The DLE model described here is designed for periodically sampled data (with sample size T), although a similar model for non-periodic data could easily be constructed.

The value of d_{max} is an adjustable parameter. For example, based on the previous IDA results (see Figure 5.1), the value of d_{max} should be to set to the crossover point of the linear and constant extrapolation models.

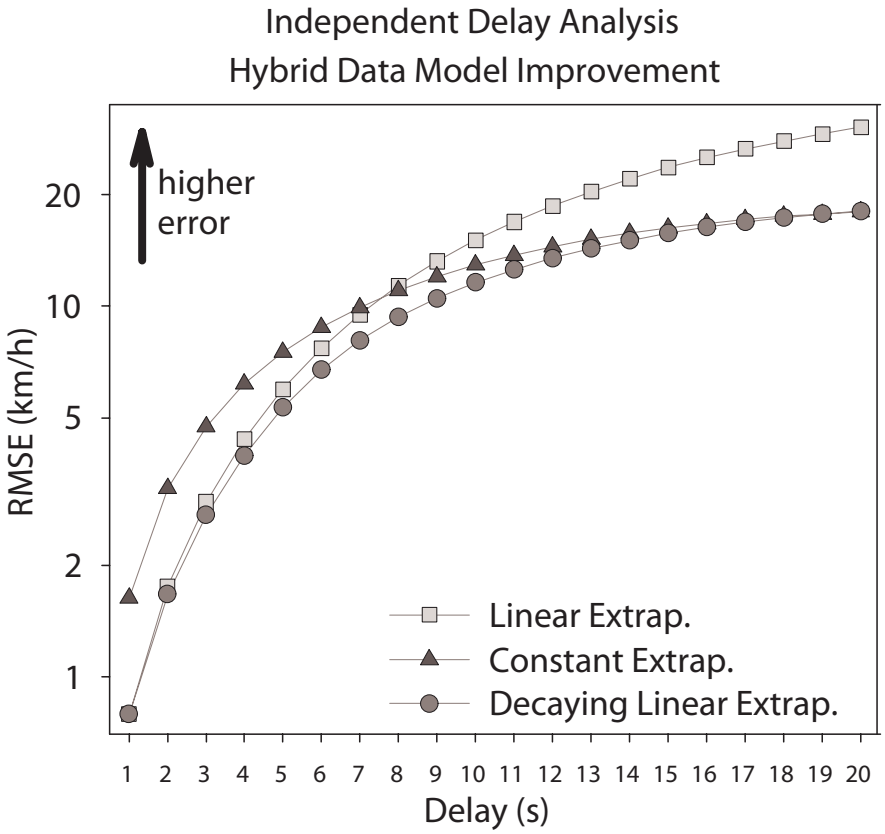


Figure 5.2. The decaying linear extrapolation model is better than the error performance of the linear and constant extrapolation models.

Now we use the IDA method on the new model and compare its results to the previous ones. Figure 5.2 shows the IDA results for the constant extrapolation, linear extrapolation, and decaying linear extrapolation models. The DLE model is actually better than both of the original models for delays less than 15 seconds, and it is comparable to the constant model thereafter.

In order to validate these results, abstract network simulations were run with various DLE models. The results are shown in Section 6.2.2, specifically Table 6.1. Although the MSE performance improvement is modest, the simulation results agree with the insights offered by the IDA and demonstrate the usefulness of the technique in 1) evaluating and

selecting from among existing data models and 2) offering insight into new models with improved performance.

We have mentioned two combined models: the piecewise model and the DLE model. In the example, the DLE model performed better than the piecewise model for this data set, but that will not necessarily be the case for other data sets. If combining models in a more sophisticated way results in higher error, then the best thing to do is fall back to the piecewise combined model. There are many other ways two or more models could be combined. For example, a combined model based on two more complex models could use weighted averages near the crossover point to create a smooth transition from one model to the next [44]. We could evaluate other combined models, but the results would still be specific to this driving data application. Our purpose here is to demonstrate the application of our technique.

In any case, model selection is still a design problem. Choosing appropriate models for the original IDA evaluation requires a certain amount of insight into the data and the models themselves. Similar insight may be needed to go beyond the piecewise combined model and create an appropriate combined model. The important idea is this: whatever models a system designer can conceive of, the IDA analysis can be used to compare them and allow the designer to choose the right models for a particular application.

5.4 Summary

In this chapter, we describe Independent Delay Analysis, a method for examining the performance of data models for filter underflow policies. This method applies an input data set to each model with a fixed delay and computes the mean squared error between the input and the estimates produced by the model. Although similar in principle to simulating a

network with a fixed delay, this method can be executed with a numerical analysis toolkit such as Matlab [45]. The method is fully described in Section 5.1.

The results from the IDA are useful because they show the behavior of a model on a particular data set for a range of delays, independent of any network characteristics. To evaluate models for a particular application, we need only look at the area of the IDA graphs where the delays correspond to the delays expected on the network being used, as described in Section 5.2.

Section 5.3 shows how insights from the IDA can be used to develop new model policies. The Decayed Linear Extrapolation filter underflow policy is developed here, which is among the policies compared in the simulations in Chapters 6 and 7.

Chapter 6

Abstract Network Simulation Results

To compare the performance of mechanisms in a general sense, we simulate a system that models the enterprise to embedded scenario using abstract network models. The enterprise network is modeled with a Poisson arrival process, and the embedded network is modeled with a periodic service process. Various mechanisms are included, and their performance is compared by measuring the input-output error using the mean squared error metric. This simulation setup is described in greater detail in Chapter 3.

This chapter presents the results of the abstract network evaluations. Policies for each mechanism are examined, and the results for various mechanisms are compared to each other. Many of the results presented below use box plots to summarize the results for a particular metric for a single experiment. The statistics given in a box plot are summarized in Figure 6.1. The results from a set of experiments are presented in a single graph to facilitate comparison of the results.

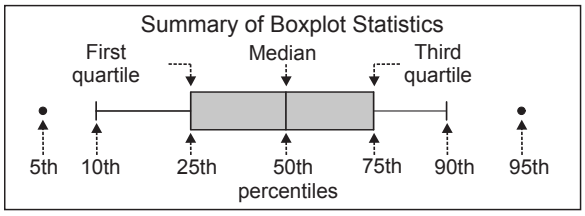


Figure 6.1. Summary of the Box Plot Diagram

6.1 Queue Mechanism Results

This section describes the policies that are applied to queue mechanisms in the abstract network simulations and highlights some of the significant results. We examine unbounded queues, bounded queues of varying lengths, and various drop policies for the bounded queues.

Although experiments were performed on all four of the input data sets described in Section 3.3, only a selection of results are included in this chapter. The results from the other data sets are qualitatively similar and are included in Appendix A.

6.1.1 Unbounded Queues

The experiments described here use an unbounded queue mechanism. Figure 6.2 shows a selection of time series data from a single trial of the experiment. Part (a) shows the input and output data streams and highlights the delay between the input and output. Part (b) shows the size of the queue over time. The delay increases as the queue length increases. As might be expected, the delay is directly proportional to the queue length, since the length of the queue when a message arrives determines how long it remains in the queue.

One goal is to see how long the queues can grow. Figure 6.3 shows the distribution of maximum queue lengths for this experiment. While the median of the distribution is around 38, the maximum queue length observed is 125. Although longer queues are less likely, there is no theoretical upper bound on the worst-case queue length for an infinitely-long data set.

These experiments show that transient queue lengths and delays can grow quite large, even if the average rate of the data going in and out of the queue is the same. While this

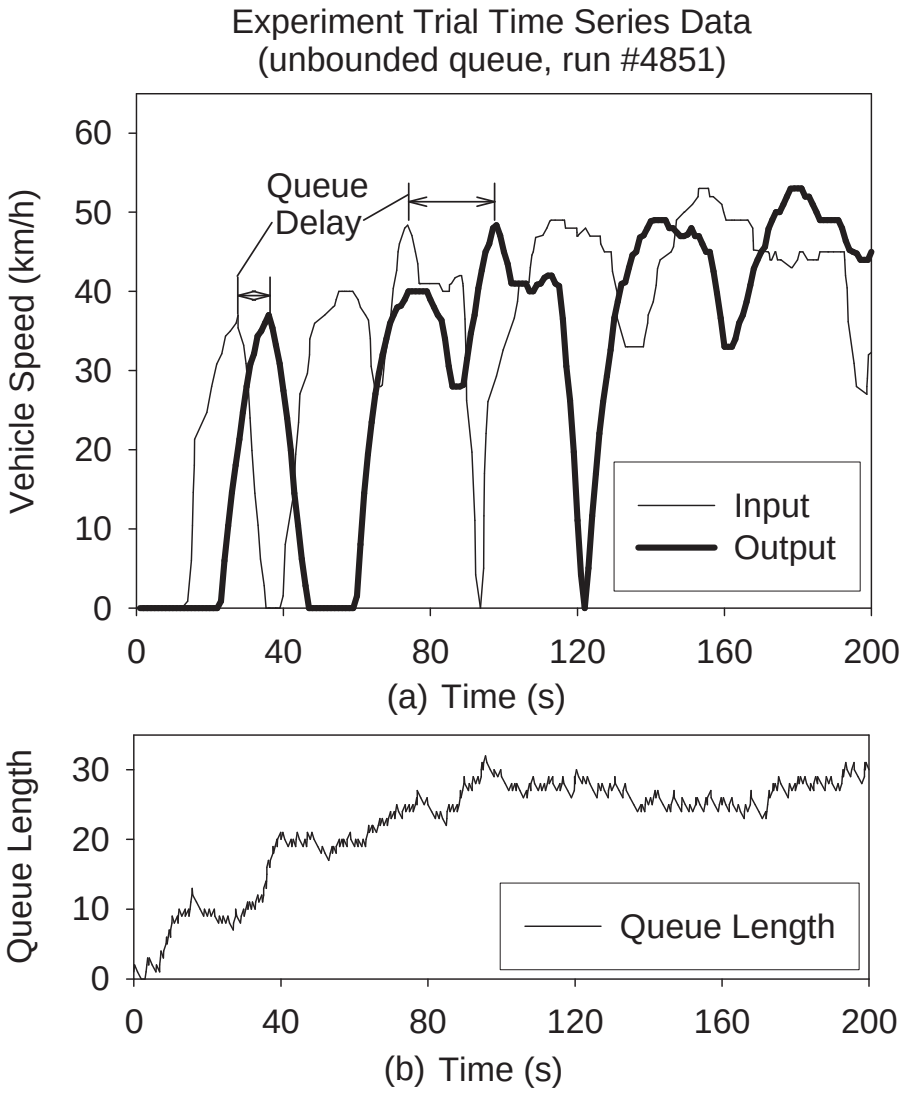


Figure 6.2. Time series data from a single trial of a queue mechanism experiment. Queue delay increases as the queue length increases.

result may be expected, it is important because it leads to the examination of bounded queues as a way to mitigate this delay.

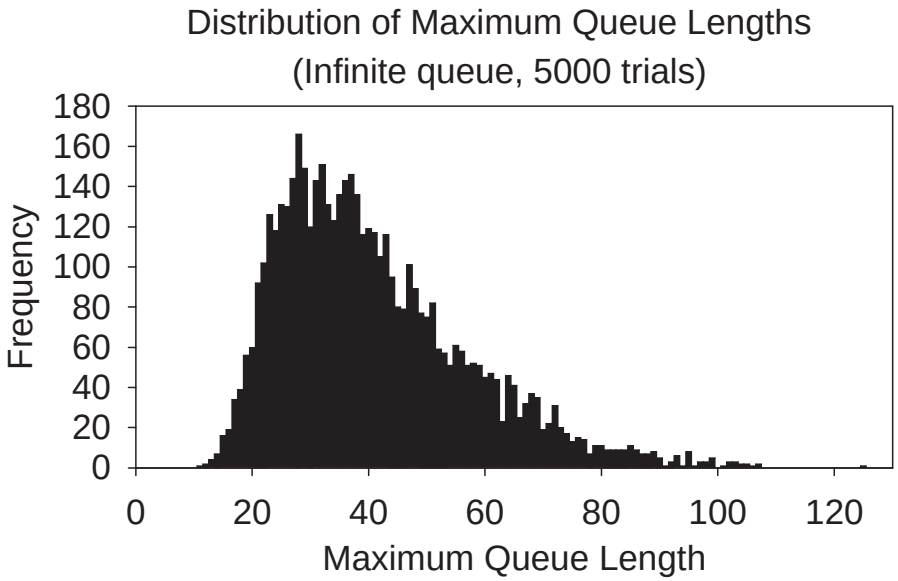


Figure 6.3. Distribution of maximum queue lengths observed during each of 5,000 trials.

6.1.2 Comparison of Queue Length Policies

We now examine bounded queues, since a bounded queue should have bounded delay. For bounded queues, there are two parameters to consider: the length of the queue and the overflow policy. The underflow policy used is always the mailbox policy.

First, we examine the effect of queue length on delay and on the number of dropped messages. Figure 6.4 shows that the number of dropped messages decreases as the queue length increases. For queues of length 50 or more, very few messages are dropped at all because longer queues are less likely to overflow.

Figure 6.5 shows that the average delay increases as the queue size increases. For queues of length 40 or longer, the median value of the average delay begins to level off, although the upper bound on delay continues to grow. Just as longer queues are less likely to overflow, they are also less likely to be full, which means that as queue bounds become larger,

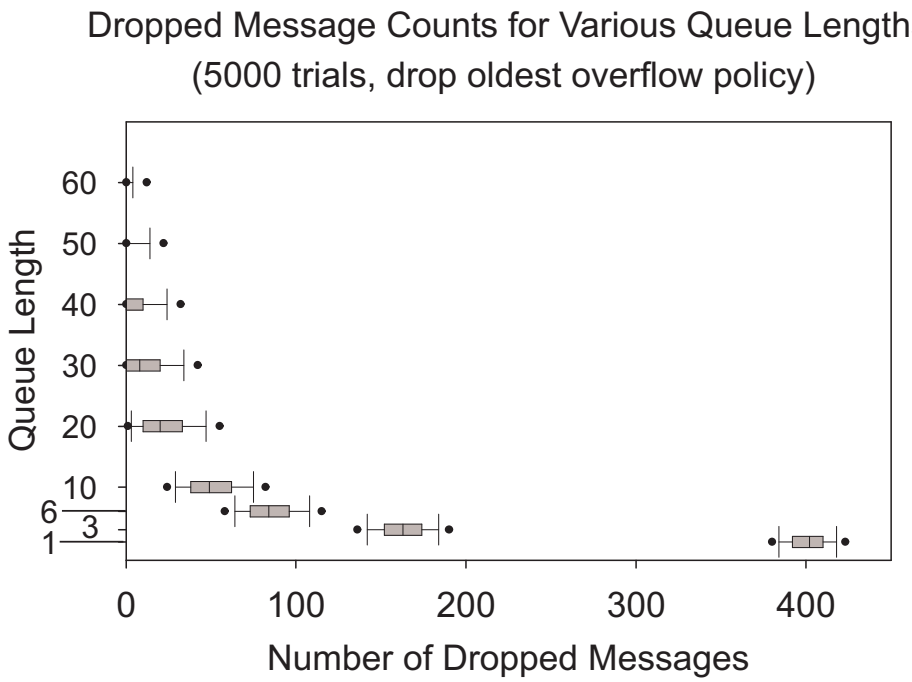


Figure 6.4. Summary of the total number of dropped messages from each trial for experiments with queues of various lengths.

the median delay is governed less by the length of the queue and more by the timing of the arrival messages. Recall that each experiment uses the same set of arrival sequences.

Based on the results in Figures 6.4 and 6.5, we observe that there is a trade-off between the number of dropped messages and the average queue delay.

Now we examine the effect of queue length on the mean squared error (MSE). Recall that the MSE metric produces a single value for each trial in the experiment. The box plots in Figures 6.6 and 6.7 compare the MSE for experiments run with bounded queues of various lengths using the Drop Oldest overflow policy.

One might expect that reducing queue delay would also decrease the MSE in the output. Indeed, the results in Figure 6.6 show that median MSE does go down slightly as the queue length is reduced. However, we also observe that the 5^{th} percentile of the error actually

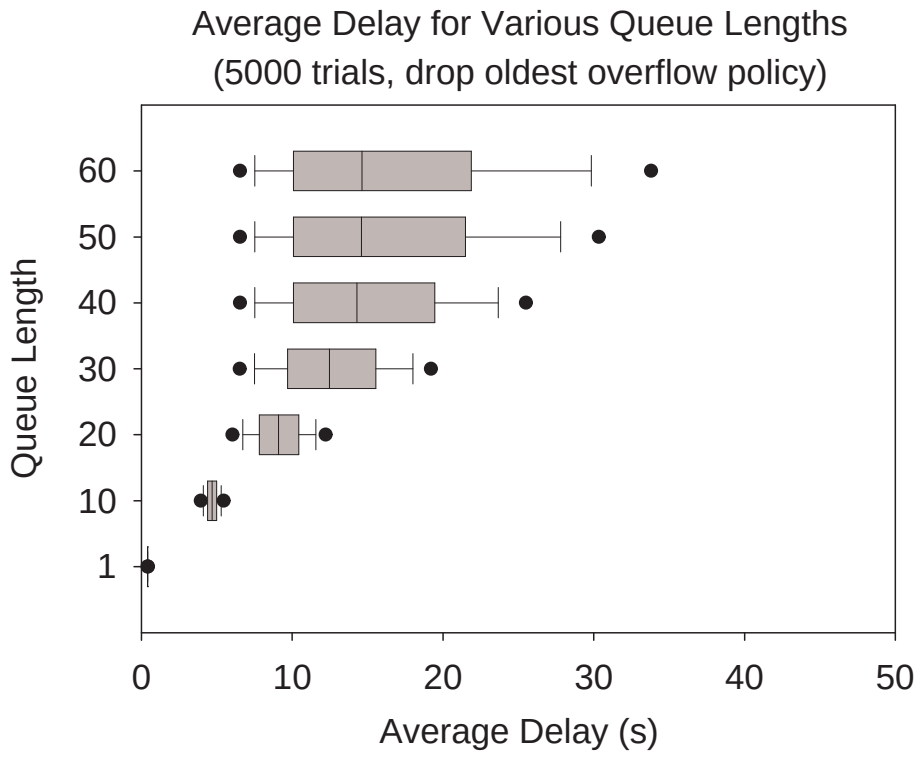


Figure 6.5. Summary of the average queue delay for queues of various lengths.

increases for short queues. Although the delay has been reduced by reducing the queue size, the shorter queues drop more messages, and these dropped messages also contribute to the MSE. By contrast, in Figure 6.7, the median error is actually greater for shorter queues. These results in Figures 6.6 and 6.7 were selected to show that the queue length parameter has an inconsistent effect on the error. The complete results are given in Appendix A.

6.1.3 Comparison of Queue Overflow Policies

Figure 6.8 shows a comparison of experiments using a length 50 bounded queue with various overflow policies. The performance of the Drop Newest, Drop Oldest, and Drop Random policies is almost the same because when an overflow condition occurs, each policy

drops a single value. Although each policy selects a different message to drop, the number of dropped messages, and thus the overall effect, is relatively small. The only policy that exhibits different behavior is the Drop All policy. The performance of this policy is worse because the flushing of the queue results in a large number of dropped messages. There is a slight variation in the Drop Oldest policy's median value in Figure 6.8, but no larger variation is observed in all the other results from all the data sets. Similar graphs for the remaining data sets can be found in Appendix A.

The insight to be gained from these experiments is that neither queue length nor overflow policy can significantly improve the mean squared error performance of the gateway.

6.2 Filter Mechanism Results

This section presents the abstract network evaluation for filter mechanisms under various model policies. We examine linear, quadratic, and cubic extrapolation models as well as the Decaying Linear Extrapolation (DLE) model. Results from the Piecewise Combined model are omitted because they are outperformed by the DLE model in every case. These model policies are described more fully in Section 4.2.1.

6.2.1 Comparison of Queue and Filter Mechansims

To compare the filter and queue mechanisms, we evaluate them using the mean squared error metric. Figure 6.9 shows the mean squared error summary comparing the constant extrapolation filter mechanism to queue mechanisms with several different length policies. The two bounded queues in the results use the Drop Oldest overflow policy. Varying the overflow policy does not significantly affect the results. Although the 95^{th} percentile of the mean squared error is only slightly lower, the filter mechanism shows a bias toward

Table 6.1. Median MSE for Various Filter Mechanisms

Mechanism	Beechwood	Squirrel Hill	Monroeville I	Monroeville II
Filter(Extrap, 0)	119.82	116.35	15.82	22.70
Filter(Extrap, 1)	219.96	271.35	21.89	35.43
Filter(DLE, 2)	106.39	109.46	12.49	12.49
Filter(DLE, 4)	105.27	109.21	12.39	12.39
Filter(DLE, 6)	107.45	113.90	13.59	13.59
Filter(DLE, 8)	114.02	121.35	15.97	15.97
Filter(DLE, 10)	123.96	132.68	19.51	19.51
Filter(DLE, 12)	138.04	145.55	24.38	24.38
Filter(DLE, 14)	154.09	163.88	30.20	30.20

lower mean square errors. One of the goals of this work is to show that application-aware mechanisms can improve performance over generic mechanisms. These results show that the Constant Extrapolating filter mechanism has lower MSE than the various queue mechanisms.

6.2.2 Comparison of Filter Mechanisms

This section compares the performance of filter mechanisms with various underflow policies. Figure 6.10 and Table 6.1 present the abstract network results for the Beechwood data set. There are two important results here. First, the DLE mechanisms show a modest improvement in median MSE over the generic Constant Extrapolating Filter. Second, the median MSE of the DLE mechanisms is better for thresholds up to eight seconds, which is consistent with the breakpoint shown in the Independent Delay Analysis conducted in Section 5.3.

6.3 Summary

This chapter describes the results from simulating the abstract network models from Chapter 3. Both queue and filter mechanisms are studied using various policies described in Chapter 4.

In Section 6.1, we compare the performance of policies for queue mechanisms and observe that changing the queue length policy does affect the error performance, but not in a systematic way — shorter queues are better for some data sets, and longer queues for others. However, the queue overflow policy — choosing which message is dropped when the length is exceeded — has relatively little effect on performance. The only overflow policy that performs differently is the Drop All policy, and it is usually worse than the others.

Queues (generic mechanisms) are compared to filters (application-aware mechanisms), and we show that filter mechanisms not only improve the median error performance, but also show a distribution with a markedly different character. These results are shown in Section 6.2.1.

Finally, the performance of different filter underflow policies is evaluated in Section 6.2.2. In this evaluation, certain hybrid and extrapolating underflow policies have better performance than the basic (constant-order) extrapolating model underflow policy. Furthermore, these improvements are consistent with the Independent Delay Analysis conducted in Chapter 5.

These results show both (a) how policies for a single mechanism cause performance to vary and (b) how application-aware mechanisms can provide better performance than generic mechanisms. These results are consistent with the results from the traffic control case study in Chapter 7.

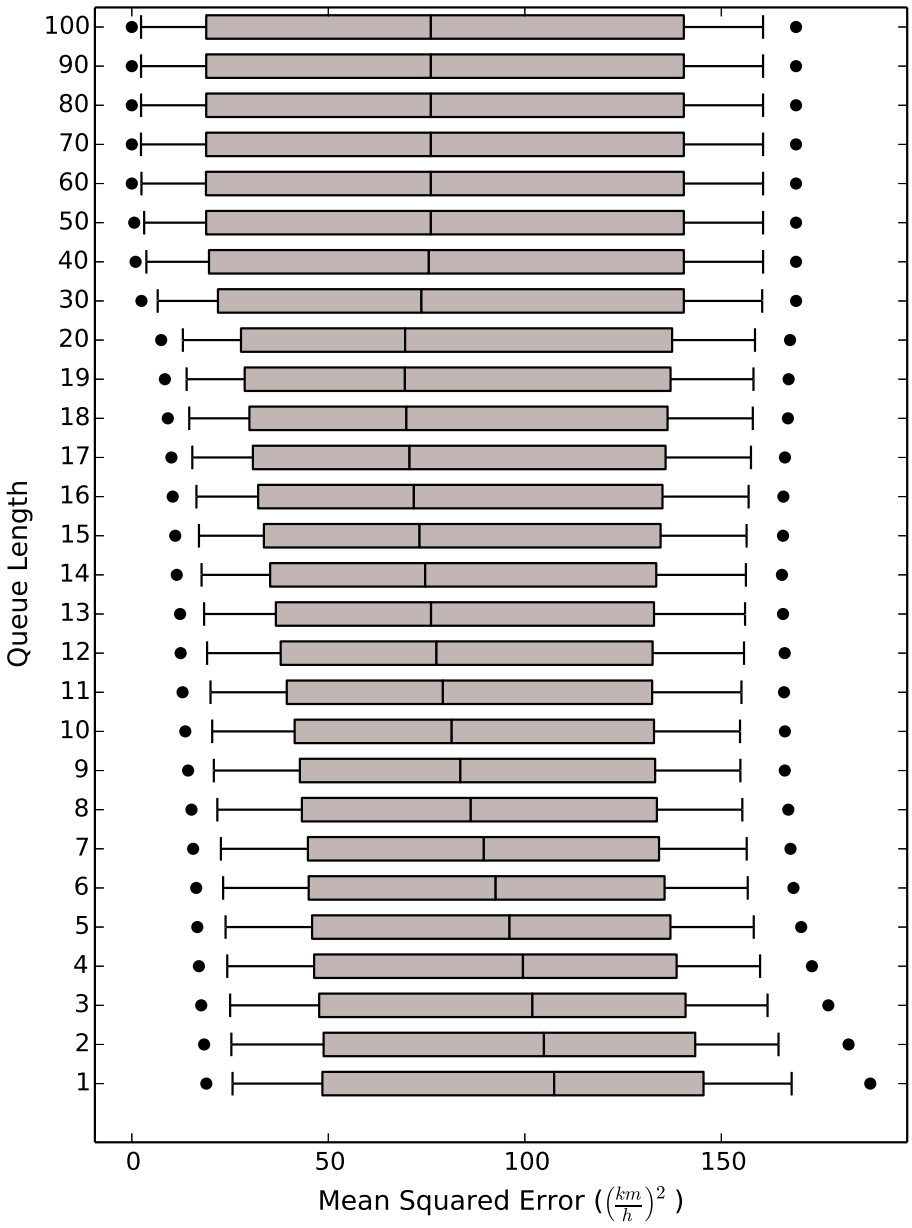


Figure 6.6. MSE for Various Queue Lengths (Monroeville I Data Set) — shorter queue lengths have a higher median value for mean squared error.

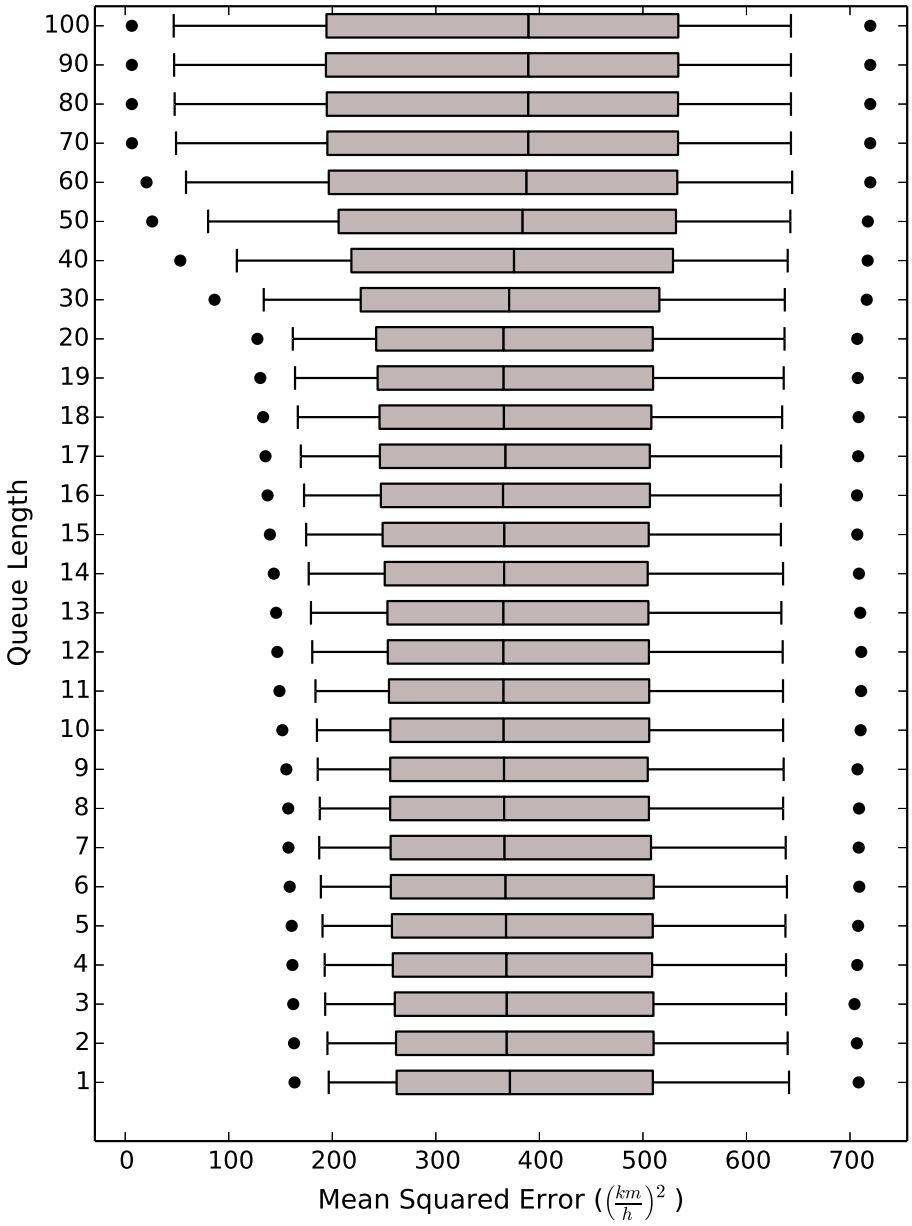


Figure 6.7. MSE for Various Queue Lengths (Beechwood Data Set) — shorter queue lengths have a slightly lower median value for mean squared error.

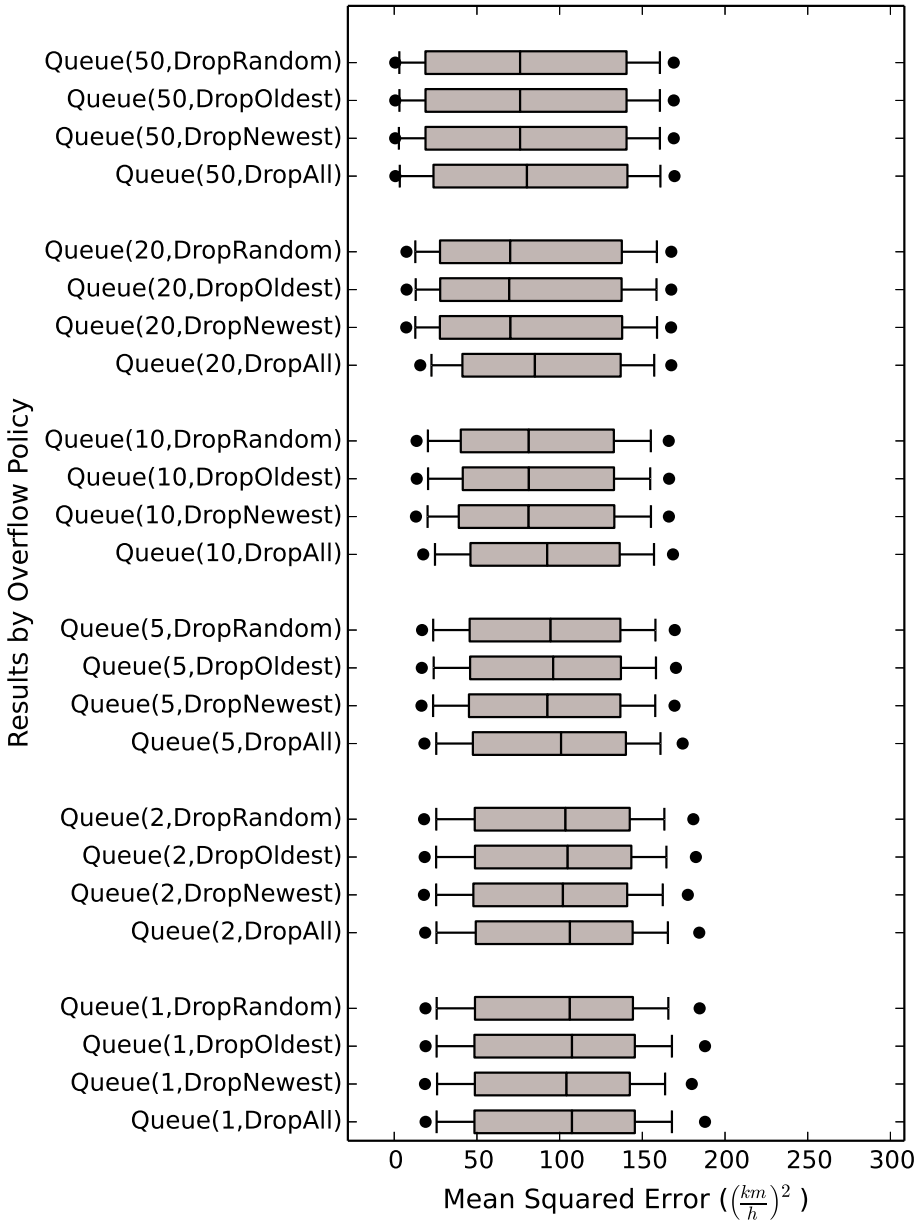


Figure 6.8. MSE for Various Overflow Policies Grouped by Queue Length (Monroeville I Data Set) — summary of mean squared error for queues of various lengths employing various overflow policies. There is little difference in the performance of the policies, regardless of queue length.

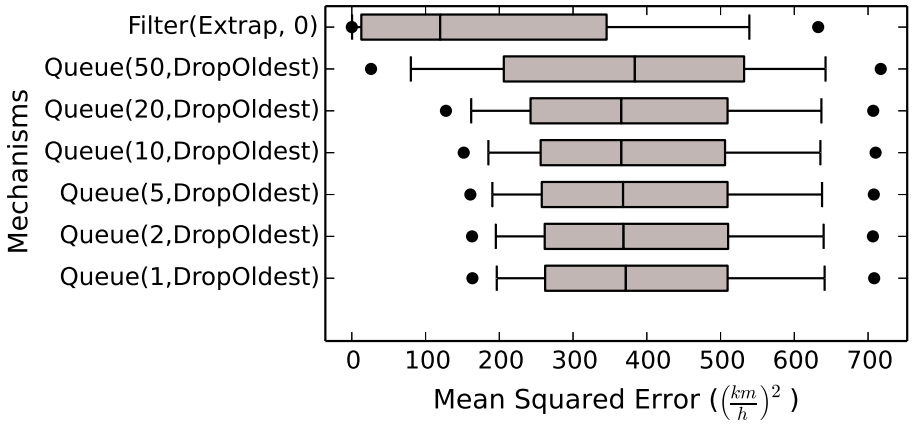


Figure 6.9. Comparison of Filter and Queue Mechanisms — the filter mechanism shows an improvement in mean squared error over queue mechanisms.

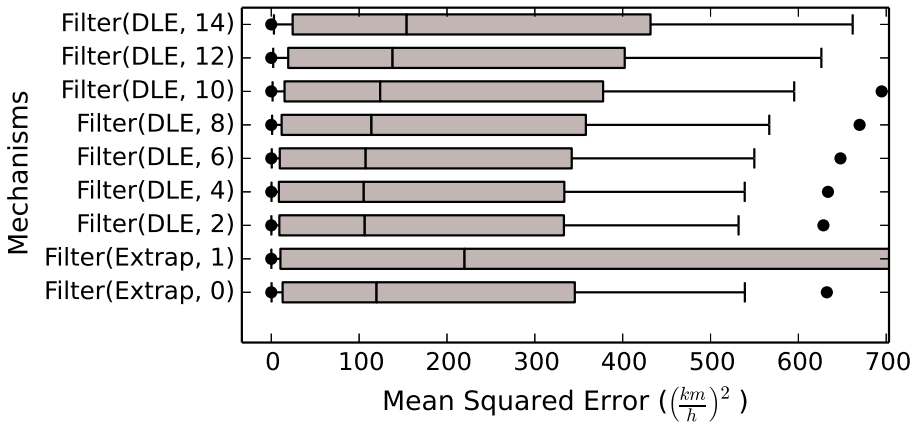


Figure 6.10. Filter Mechanism Comparison for Beechwood Data

Chapter 7

Traffic Control Case Study

In order to evaluate the performance of various gateway mechanisms in the context of an application, we present a case study of a traffic control application. A central control node issues guidance messages to cars on a road to help them adjust their speed so that they arrive at the light when it is green. The goal of the application is to reduce the number of cars stopped at the light and to provide a smoother flow of traffic.

We adapt this application to the enterprise-to-embedded gateway scenario by transmitting the guidance information from the control node to each vehicle node over an enterprise network (including a wireless network). Each vehicle contains a gateway that connects the enterprise network to the on-board real-time network. The guidance packet passes through the gateway, is transmitted on the vehicle's real-time network, and then reaches the vehicle controller that adjusts the acceleration of the car to attain the guidance speed.

For a case study, we adapt the simulation described in [8], which consists of a fixed number of vehicles traveling around a ring road with a single stoplight. Cars are not allowed to pass, and no cars enter or exit the road during the simulation. Cars pass the light when it is green, but stop when it is red. Although it is simple, the ring road scenario is commonly used for evaluating vehicle dynamics [33]. In order to improve the flow of traffic, a central node computes guidance recommendations to speed up or slow down cars so that they will reach the stoplight when it is green. The objective is to smooth out traffic flow and

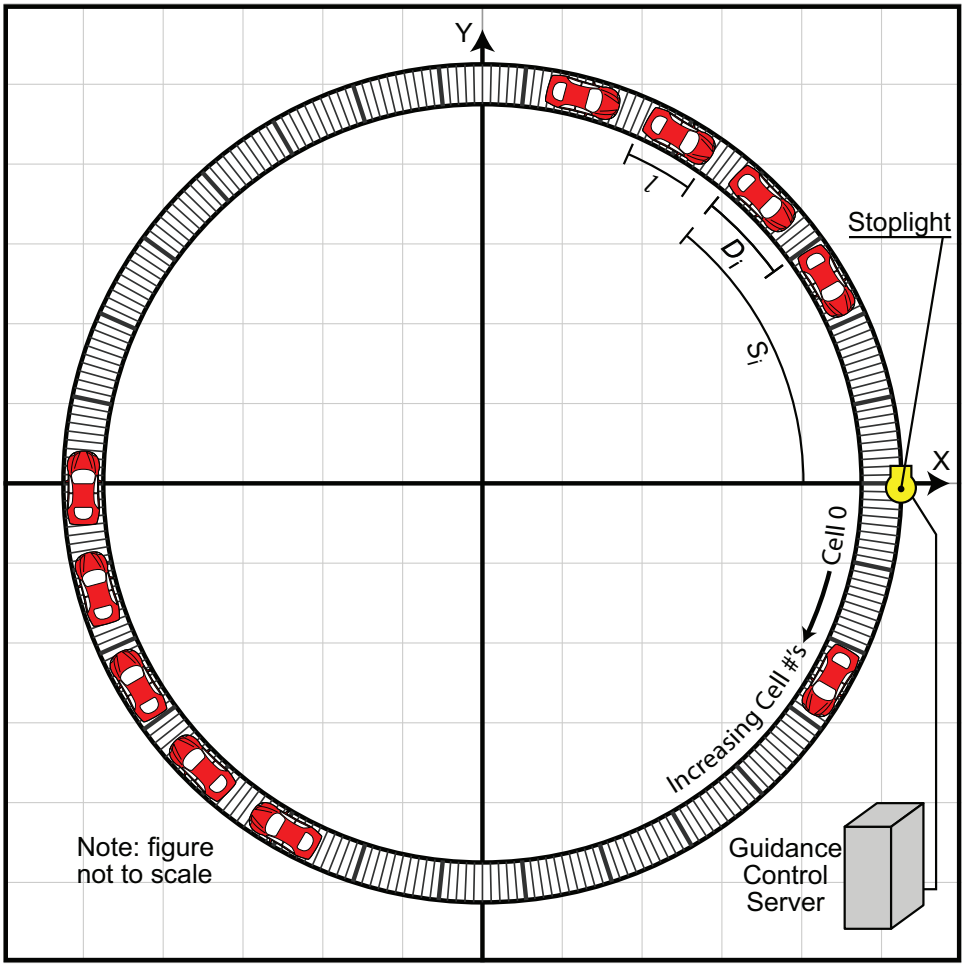


Figure 7.1. Overview of the simulation used in the case study.

reduce the number of cars caught at the red light. The physical arrangement of nodes in the simulator is shown in Figure 7.1

The motion of the cars in the simulation is modeled using cellular automata (CA), a microsimulation approach to traffic modeling. Microsimulation is preferred to macrosimulation because it provides a way to model each vehicle in the system individually so that each car can receive and act on guidance provided by the central traffic controller.

The case study is useful because it demonstrates the impact of a non-real-time network on the performance of a real-time system. The application is simple, but provides good

application-level metrics (see Section 7.2). It is also fairly straightforward to implement the CA traffic simulation in the OPNET® Modeler network simulation to observe the interactions and network behaviors of the system.

The remainder of this chapter describes the simulation configuration in greater detail. In Section 7.1, we describe the traffic control algorithm and vehicle models from [8]. In Section 7.2, we describe the metrics that are computed during the simulation runs. In Section 7.3, we describe the implementation of the application in OPNET® Modeler including the design of various network nodes. In Section 7.4, we describe the various parameters that can be adjusted in the simulation, including parameters related to various wired and wireless network models. In Section 7.5, we discuss a selection of the traffic simulation results. The complete results are listed in Appendix B.

7.1 Traffic Control Algorithm

The traffic control algorithm used for the case study is published in [8]. The details are reproduced here, including slight changes made to adapt the algorithm to the simulation application.

7.1.1 Parameters and Variables

The following parameters are defined for the algorithm. They are fixed for any particular simulation.

- N is the number of vehicles in the simulation.
- L is the length of the ring road in cells.
- l is the size of the vehicle in cells.

- $\rho = \frac{l \cdot N}{L}$ is the overall density of traffic in the simulation.
- D_0 is the position (cell number) of the stoplight.
- v_{max} is the maximum velocity (cells/tick) allowed for vehicles in the simulation.
- a_{max} is the maximum positive acceleration (cells/tick/tick) allowed for vehicles in the simulation.
- p is the probability of random deceleration in the CA model.

a_{max} is defined in [8] as a, but is also implicitly assigned a value of 1 in the description of the CA model. In both our model and that in [8], the maximum negative acceleration is effectively infinite because the vehicle models are constrained to avoid colliding with other vehicles or passing a red light. It is consistent with real-world vehicles in the sense that maximum braking deceleration is typically much greater than maximum acceleration.

The following variables are defined for the algorithm.

- t is the current time.
- x_i is the current position (cell number) of the rear of the i^{th} vehicle. This value is modified by the CA model during simulation.
- v_i is the current velocity (cells/s) of the i^{th} vehicle. This value is modified by the CA model during simulation.
- v_i^d is the current desired velocity (cells/s) computed by the guidance algorithm and transmitted to each vehicle.
- $d_i = x_{i-1} - x_i - l$ is the number of empty cells between the current vehicle and the following vehicle.

- $r_i = D_0 - x_i - l$ is the number of cells to the stoplight. In [8], this value is assigned the symbol s_i, but we use r_i to avoid confusion with the symbol s_i used the description of the filter underflow policy models in Chapter 4.

- N_0 is the number of vehicles where $v_i = 0$.

7.1.2 Computing Velocity Guidance

The velocity guidance algorithm in [8] identifies the fastest velocity at which a car can reach the light so that the light will be green when the car passes. We use this algorithm for the case study, with a slight modification. In [8], vehicles receive a guidance update once per circuit when they pass a fixed point. In this application, the algorithm is used to compute guidance values on an ongoing basis and the results are transmitted to the vehicles periodically. Although the presentation here is different (for the sake of clarity), the algorithm used here is equivalent to the one presented in [8].

First, the function describing the behavior of the stop light is defined as:

$$\text{lightstate}(t) = \begin{cases} \text{green} & \text{if the light will be green at time t} \\ \text{red} & \text{otherwise} \end{cases} \tag{7.1}$$

The function "arrival" is defined in Equation 7.2. It extrapolates the future motion of each vehicle to predict the future time when the vehicle will pass the light given its current speed, the distance to the light, and a candidate value for velocity guidance v_t.

$$\text{arrival}(v_i, r_i, v_t) = \left(t + \frac{|v_i - v_t|}{a_{max}} + \frac{r_i - s_a + l}{v_t} \right) \tag{7.2}$$

The second term is the time it takes the vehicle to adjust from v_i to v_t, and the third term is the time it takes to reach the light at the new velocity v_t. s_a is the distance traveled during the adjustment period, defined as:

$$s_a = \frac{(v_i + v_t)|v_i - v_t|}{2a_{max}} \tag{7.3}$$

Using the functions defined in Equations 7.1 and 7.2, the velocity guidance can be defined as:

$$v_i^d = \max\left(v_t \in [1,2,\ldots,v_{max}]\right) \mid \text{lightstate}\left(\text{arrival}(v_i, r_i, v_t)\right) \rightarrow \text{green} \tag{7.4}$$

7.1.3 Cellular Automata Model

The cellular automata model described here is the same as the one presented in [8]. The steps of the model are executed ten times per second. The models are executed using parallel dynamics [46]. That is, each step is executed for every CA model in the simulation before any model executes the following step.

The steps of the CA model are:

1. **Accelerate:** (attempt to reach guidance velocity)

$$v_i \rightarrow \begin{cases} \min(v_i + a_{max}, v_i^d) & if\, v_i < v_i^d \\ \max(v_i - a_{max}, v_i^d) & if\, v_i > v_i^d \\ v_i^d & otherwise \end{cases}$$

2. **Decelerate:** (because of blocking by the car in front or the stoplight)

$$v_i \rightarrow \begin{cases} \min(v_i, d_i, r_i) & \text{if lightstate}(t_{current}) = \text{red} \\ \min(v_i, d_i) & \text{if lightstate}(t_{current}) = \text{green} \end{cases}$$

3. **Randomization:**

   With probability p, $v_i \rightarrow \min(v_i - 1, 0)$

4. **Movement:**

   $x_i \rightarrow (x_i + v_i) \mod L$

5. **Guidance Update:**

   In this step, the stoplight node updates its guidance model. Including the guidance update in the CA model ensures that the guidance update occurs after the CA model has advanced. Once the guidance update occurs, the new guidance values are passed to the network (non-CA) portion of the simulation, as described in Section 7.3.

7.2 Application Metrics

This section describes the metrics that are relevant to the traffic application. These metrics will be used to compare the performance of the gateway mechanisms. In [8], two metrics for measuring the performance of the traffic simulation are defined: motionless ratio and flow. We also define two additional metrics based on fuel consumption models.

7.2.1 Motionless Ratio

The ratio of motionless vehicles, r_0, is defined as the fraction of vehicles in the simulation that are stopped at a given time:

$$r_0 = \frac{N_0}{N} \tag{7.5}$$

where N is the number of vehicles in the simulation, and N_0 is the number of vehicles whose velocity is zero.

7.2.2 Flow

The second metric given in [8] is traffic flow, q, which is defined as:

$$q = \frac{\rho}{N} \sum_{i=1}^{i=N} v_i = \frac{l}{L} \sum_{i=1}^{i=N} v_i \tag{7.6}$$

where N is the number of vehicles in the simulation, ρ is the overall density of traffic in the simulation, l the length of a vehicle in cells, L the length of the road in cells, and v_i is the velocity (in cells/tick) of the i^{th} vehicle in the simulation.

7.2.3 Fuel Consumption

The discussion of results in [8] suggests that the goal should be "stable flow," although this term is not rigorously defined. It is reasonable to expect a "good" traffic control algorithm to reduce variations in speed as much as possible. For this reason, we have selected an additional metric: fuel consumption. The value of a fuel consumption metric is that fuel consumption models depend on both the velocity and acceleration of vehicles (while the flow metric from [8] depends only on velocity).

We have selected two fuel consumption metrics based on models described in [47] and [48]. These models are chosen from the literature because they are widely cited, they are validated with empirical results, they provide models that are compatible with the simulation data, and they provide model parameters for an average vehicle. Although both are based on instantaneous speed and acceleration, the energy-related model in [47] includes more high-order terms and is based on direct measurement of fuel consumption, while the aggregate model provided in [48] is a simpler model with values based on measurement of emission gases.

7.2.3.1 Energy-based Model of Fuel Consumption

In [47], the authors provide an Energy-based Fuel Consumption (EFC) model based on an analysis of the energy required to maintain engine operation, to move the vehicle forward, and to overcome drag. The model is validated using a fuel flow meter to measure consumption on two vehicles over a variety of driving scenarios, and the model has less than a 12% mean error in all cases and less than a 6% mean error for common driving scenarios.

The expression for instantaneous fuel consumption defined by the model is given as:

$$f_t = \begin{cases} \alpha + \beta_1 R_t v + \left[\frac{\beta_2 M v a^2}{1000}\right]_{a>0} & R_t \geq 0 \\ \alpha & R_t < 0 \end{cases} \tag{7.7}$$

where R_t, the total tractive force, is given by:

$$R_t = b_1 + b_2 v^2 + Ma/1000 \tag{7.8}$$

In [47], Equations 7.7 and 7.8 are given in terms of a_e, the effective acceleration due to the vehicle acceleration and acceleration due to gravity. We have reduced this value to a, the vehicle acceleration, since the simulation assumes a road with 0% grade. For our computations, we use the default parameter values given in [47] and reproduced in Table 7.1.

7.2.3.2 Aggregate Fuel Consumption Model

A second fuel consumption metric is based on [48], which describes the fuel consumption characteristics of light duty vehicles and is designed for use with traffic microsimulation. It uses an estimator of vehicle power demand called **vehicle specific power** (VSP) to develop

Table 7.1. Default Parameters for Instantaneous Fuel Consumption Model from [47]

Parameter	Value	Description
α	0.444	Idle fuel rate in mL/s
M	1200	mass in kg
β_1	0.090	Energy efficiency in ml/kJ
β_2	0.030	Energy-acceleration efficiency in $\frac{mL}{kJ \cdot m/s^2}$
b_1	0.333	Drag force in kN, mainly related to rolling drag
b_2	0.0008	Drag force in $\frac{kN}{m/s^2}$, mainly related to aerodynamic resistance

a piecewise linear model of instantaneous fuel consumption called **normalized fuel rate** (NFR).

It uses empirical data to relate fuel consumption to vehicle specific power, a measure of power demand that is, in turn, based on the instantaneous velocity and acceleration of a vehicle. A formula for VSP, as described in [48], is first presented in [49]. The coefficients in this equation represent results of empirical modeling for typical light duty vehicles.

$$\text{VSP} = v \cdot (1.1 \cdot a + 0.132) + 0.000302 \cdot v^3 \tag{7.9}$$

To obtain a relationship between fuel consumption and VSP, the authors in [48] drive a variety of vehicles while recording vehicle emissions, instantaneous speed, and instantaneous acceleration. The emission measurement is used to compute the instantaneous fuel rate (in g/s). Instantaneous velocity and acceleration are used to compute VSP (using Equation 7.9). The fuel consumption results are normalized so that the normalized fuel rate (NFR) for each vehicle at zero velocity is 1. The NFR is a unitless quantity. Then the data are fit

to a piecewise linear model given by:

$$\text{NFR} = \begin{cases} \beta_s \cdot \text{VSP} + 1 & VSP >= 0 \\ 1 & VSP < 0 \end{cases} \tag{7.10}$$

The parameter β_s has no relation to the parameters of the Biggs model. [48] gives a general-purpose value 0.264 for β_s. This value is derived from empirical data using three separate vehicles and is representative of the fuel consumption characteristics of light duty vehicles.

7.3 Simulation Implementation

This section describes the implementation of the traffic control algorithm in OPNET® Modeler.

7.3.1 OPNET® Overview

This section provides a brief overview of the capabilities of the OPNET® simulation as they are used in the case study.

The OPNET® Modeler platform provides a discrete event simulator along with a framework for defining complex hierarchical models of system nodes. It provides a rich library of predefined models for wired and wireless communication platforms and the capability to extend the models and define new ones.

The basic structure of an OPNET® simulation is a scenario that defines the location of simulation objects in physical space and the wired communication links that connect them. Wireless links use global network objects to route packets and model channel characteristics. The fundamental activity underlying the network simulation is the creation and processing of network packets as they flow between nodes in the system [7].

Objects in the simulation are organized hierarchically. Simulation scenarios are composed of networks. Each network contains other networks (sub-nets) and node models. Node models are roughly the granularity of individual physical objects (e.g. a router or switch). A node model has attributes that can be changed to adjust its behavior. During simulation, in addition to processing packets, it may also generate outputs in the form of statistics (e.g. packet drop rate).

Node models consist of interconnected process models. Each process is defined with a state machine whose transitions can be triggered by interrupts, network packet arrivals, or changes in external parameters. Process models can have multiple inputs and outputs to route packets and other information to and from other processes. Figure 7.2 shows a workstation node model that is included with the OPNET® models package. Each square represents a process model that handles packets. In this model, the processes represent different layers in a wireless network stack, from the physical layer (wlan_port_rx_0_0 and wlan_port_tx_0_0) to the protocol layer (TCP, UDP, etc).

7.3.2 Customized OPNET® Simulation Models

Here we describe the customizations made to the OPNET® node and process models to implement the traffic simulation.

7.3.2.1 CA Executor Node Model

One important aspect of the simulation is integrating the network models with the CA traffic models in the OPNET® discrete event simulation. The network simulation is a fine-grained simulation of packet flows and processing, while the CA traffic model is updated on a discrete basis. Connecting the execution of these two models is accomplished using the CA Executor node model, a custom model developed for the case study.

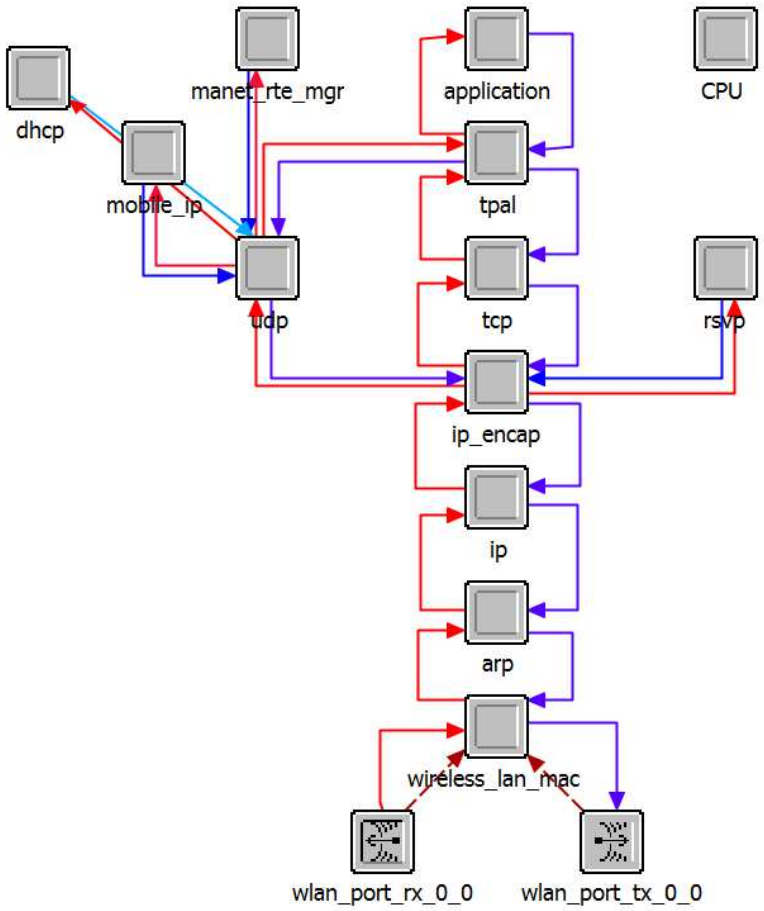


Figure 7.2. OPNET® WLAN workstation model from [7] shows the various layers of the network stack

The CA Executor node model contains a single process model, the CA Executor process model. This process module has three states: an init state, a wait state, and a state for advancing the execution of CA models in the simulation. These states are shown in Figure 7.3.

- **Init State:** The init state is executed once at the beginning of the simulation. When executed, the CA Executor identifies all node models that have a CA process model and obtains a reference to each one. These references allow the CA Executor to communicate with each CA model and advance them in lock step, and at the appropriate times in the simulation. The first **Update Interrupt** is scheduled, then the CA Executor transitions to the **Wait state**.

- **Wait State:** The CA remains idle in this state between execution interrupts. The arrival of an **Update Interrupt** causes the CA Executor to advance to the **Advance CA** state.

- **Advance CA State:** In this state, all the CA process models throughout the simulation are advanced through a complete sequence of CA steps. The order in which the models are executed is arbitrary, but the models are executed in lock step. That is, the first step is executed for all models, then the second step for all models, and so forth. This process continues until all the CA steps in all the models have been completed. Finally, the CA Executor schedules a new **Update Interrupt** for the next update period. Then the state machine transitions back to the **Wait** state.

The Update Interrupt transition is triggered by an update that the CA Executor schedules in the OPNET® simulation schedule. This interrupt occurs at a pre-determined future time in the simulation execution.

In between discrete CA executions, the network models execute according to their designed behavior, routing packets through channels and process models. Whenever a packet will affect a CA model (for example, when a guidance packet will cause a vehicle node to update its desired velocity), the packet data is stored, and the CA model acts on it during

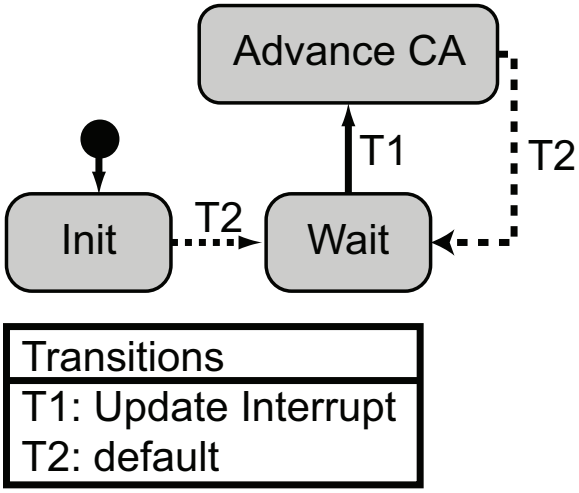


Figure 7.3. Cellular Automata Executor Process Model Statechart

its next execution. In this way, the discrete CA models and the continuous-time network models can be simulated simultaneously.

7.3.2.2 Stoplight Node Model

The basic workstation model (Figure 7.2) is augmented with several additional process models to create a stoplight node (see Figure 7.5). We have added a CA process model that models the state of the light and synchronizes the behavior of the light with the CA traffic model. We also have added the internet delay process model to model the delays of a network or networks between the CA and the wireless node. Finally, we have added the udp_tx process model to create properly addressed UDP packets.

The basic flow of information in the stoplight node model is:

1. The CA Executor causes the Stoplight CA model to execute, which includes updating the light state (red or green) and performing new guidance computations for each vehicle node.

2. The Stoplight CA process emits packets with updated guidance values for the vehicle node models (one packet per vehicle node model). At this point, the discrete CA

Sequence # (32-bit uint)	Car Index (16-bit uint)	Desired Velocity (in cells/tick) (8-bit uint)	Creation Timestamp (64-bit floating point)

Figure 7.4. Guidance packet format for the traffic control application.

execution portion of the simulation is over, and the packet traverses the networks according to the network models.

3. The internet delay process model queues guidance packets according to its delay model. When the delay model indicates the packet should be delivered, it is emitted to the udp_tx process. The delay models are described in Section 7.3.2.4. When no delay model is used, this process delivers packets instantaneously.

4. The udp_tx process model wraps the guidance packet in a UDP packet with the correct address and injects the UDP packet into the UDP layer of the workstation model.

5. From this point on, the default OPNET® model handles the packet like any other UDP packet, routing it to the destination node specified by the IP address.

The Stoplight CA model executes state updates in CA step five (after the four steps of the vehicle CA are complete) to ensure that the vehicles are updated in their new positions before new guidance values are computed. Providing a separate CA step for the stoplight model also ensures that the light state will not change in the middle of CA steps one through four, which could cause inconsistent behavior in the vehicle CA models.

The packet format used for guidance packets is given in Figure 7.4. The sequence number is incremented for each round of guidance packets. The creation timestamp is a simulation timestamp that is used to measure end-to-end delay.

7.3.2.3 Vehicle Node Model

The vehicle node model is also based on the basic OPNET® workstation model. The node model is shown in Figure 7.6. We have added:

- the udp_rx process model to process UDP velocity guidance packets from the stoplight node.
- the gateway node to model the gateway mechanism. The mechanism and its parameters are configurable on a per-simulation basis.
- the embedded servicer to model the embedded network and request packets from the gateway.
- the CA model to control the movement and dynamics of the vehicle nodes according to the CA model from [8].
- the bg_sender to model additional traffic. This model is described in Section 7.3.2.5.

The basic flow of information in the node model is:

1. A wireless packet containing velocity guidance information arrives at the vehicle node.
2. The protocol stack provided by the basic workstation node processes the packet to the UDP protocol layer.
3. The existing udp process model has been modified to divert traffic control packets to the udp_rx process model.
4. The udp_rx process model strips off the UDP packet and forwards a guidance control packet to the gateway process model.
5. The gateway process model can be configured to use any of the gateway mechanisms described in Chapter 4.

6. The periodic servicer process model emulates the real-time behavior of the embedded vehicle network by periodically requesting a packet from the gateway.

7. The packet requested by the servicer process is delivered to the vehicle CA process model. The CA process model uses the updated desired velocity (v_i^d) value from the control packet on the next update of the vehicle CA model.

8. The CA Executor advances the vehicle CA model (see Section 7.1.3 for details). The timing of guidance packet arrival is asynchronous with respect to the CA model update timing. The vehicle CA process model updates the physical position of the vehicle in the simulation during the Movement step of the CA model (see Section 7.1.3).

A bit-level CAN bus simulation was originally implemented in place of the **periodic servicer** process, but due to the real-time behavior of the CAN bus, the impact on the outcome of the simulations was negligible (when compared to the delays in the wireless networks), and the CAN bus simulation significantly reduced the overall speed of the simulation. Identical results can be obtained using the simplified model of the **periodic servicer** with much better overall simulation performance.

7.3.2.4 Internet Delay Model

One network element that is important to these simulations is a link that represents the delay characteristics of an Internet connection link. This model is inserted between the guidance packet computer and the wireless transmitter at the stoplight that sends the guidance packets to the vehicles, as shown in Figure 7.10. [50] describes several reasons why modeling the Internet is difficult: the large size of the system, the ongoing growth and change in the system, and the heterogeneity of topologies and protocols. While the OPNET® Modeler is well-suited to implementing a candidate network configuration and analyzing its

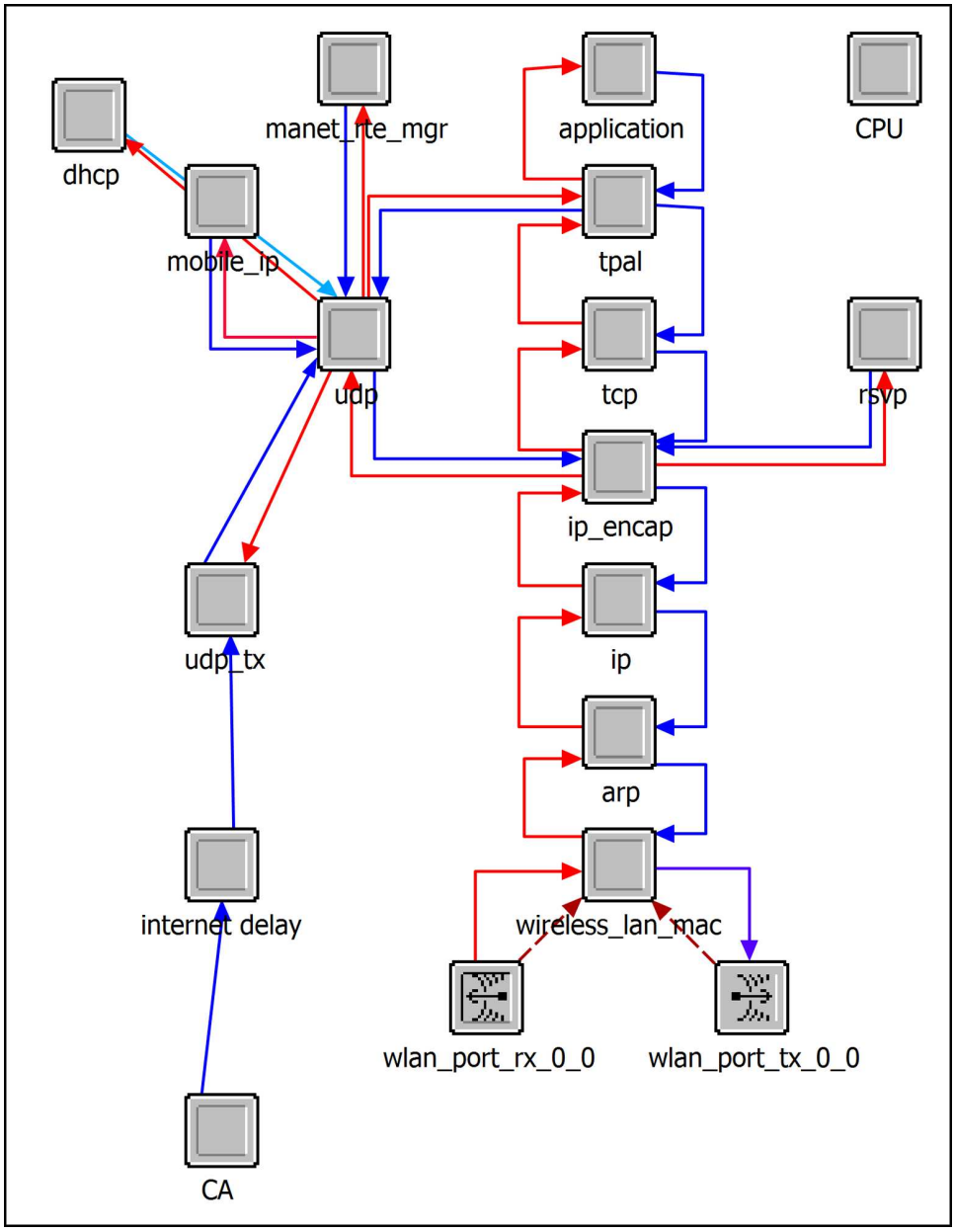


Figure 7.5. Stoplight Node Model — OPNET® WLAN workstation model modified to allow the CA stoplight model to interact with the network

behavior, developing a simulated network to represent full-scale Internet traffic behavior would involve deploying many nodes and network links, configuring them all, and then validating that configuration. The large number of configuration parameters would make it

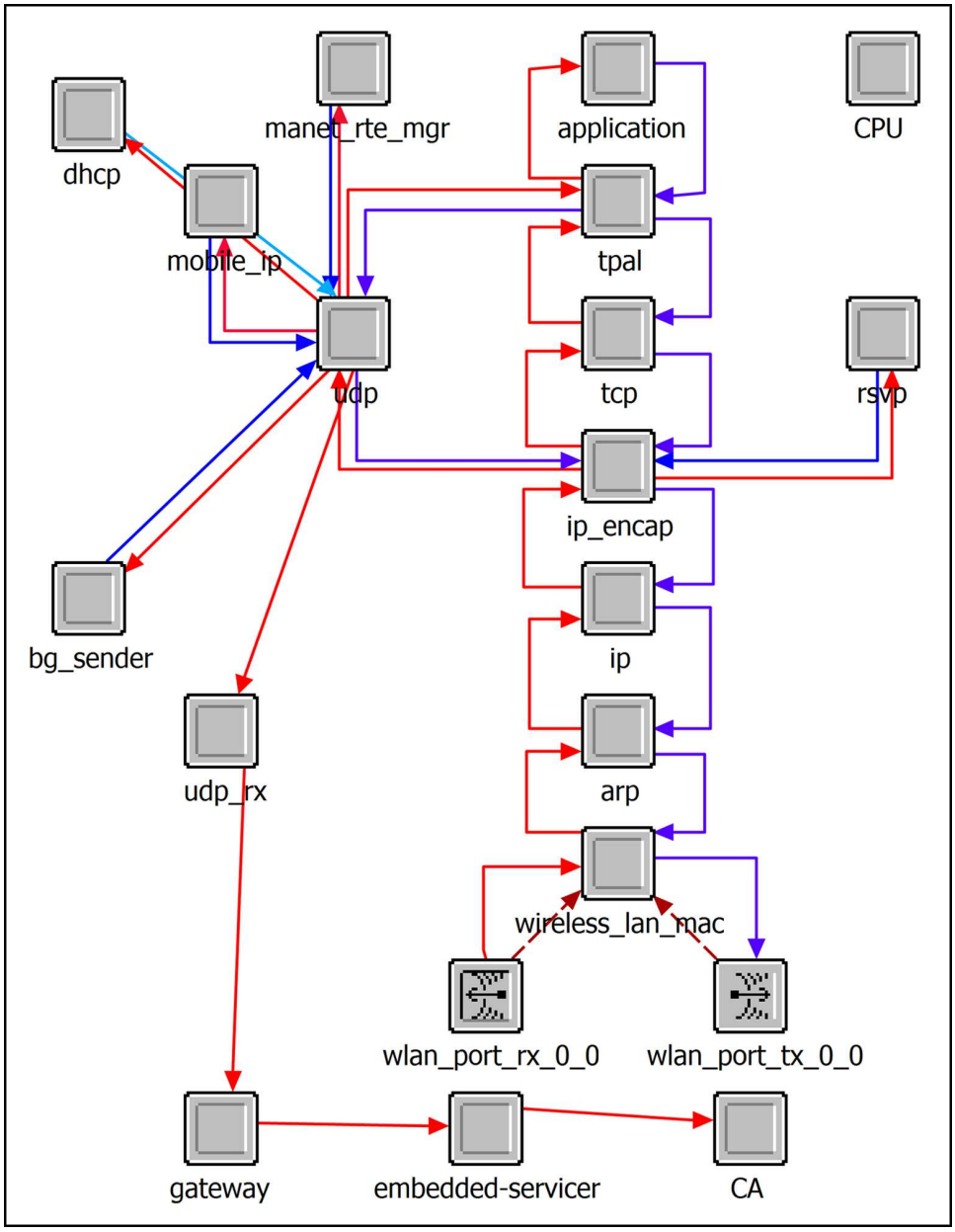


Figure 7.6. Vehicle Node Model — OPNET® WLAN workstation model modified to allow the CA vehicle model to interact with the network

difficult to determine which parameters should be adjusted to vary network behavior and observe the effect on the gateway performance. These large-scale simulations would also be extremely time-consuming to execute.

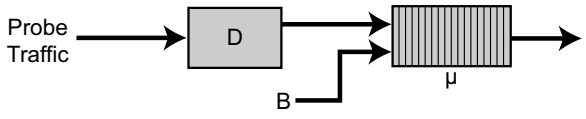


Figure 7.7. Internet Link Model from [51] models the delay behavior of periodic packets sent over the Internet.

Rather than try to implement a large-scale Internet model, we have implemented the simple model for Internet round trip time (RTT) described in [51]. There, the authors show that the RTT of periodically dispatched UDP packets is constrained by the bandwidth and utilization of the slowest link. The author describes a network link model that consists of a fixed delay component plus a single service queue used to model the variable delay component. A variable amount of traffic (representing Internet traffic) is injected into the queue between probe packet arrivals. This simple model is well-suited to this case study because the stoplight uses periodically-dispatched UDP packets to communicate velocity guidance to the vehicle nodes. The model (shown in Figure 7.7) has three parameters:

- D is the fixed packet delay in s.
- μ is the processing rate of the queue, in bits/s.
- B is a random process that determines the amount of network traffic injected between arriving probe packets.

In [51], the author uses this model to accurately describe the RTT behavior of probe packets transmitted over the Internet. One important feature that it captures is called probe compression (similar to ACK compression [52]) where groups of probe packets arrive close together.

This model is useful for modeling link behavior in the traffic simulation. First, it is a model for delivery of periodically-transmitted packets, which correspond to the periodically-

transmitted guidance packets in the simulation. Second, the model captures the compression effects of Internet traversal.

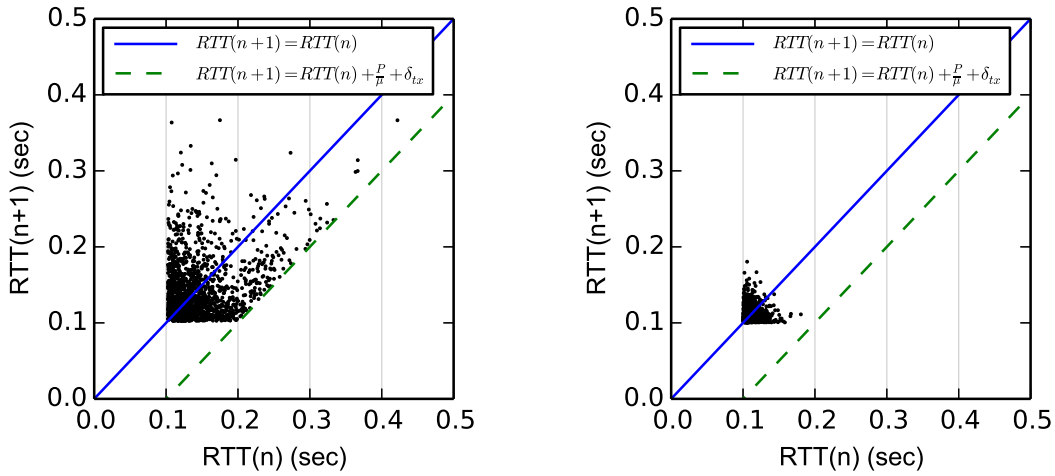
[51] also presents a method of analyzing RTT with phase diagrams. The phase diagram plots RTT(n+1) vs. RTT(n) (the round trip time of successive packets). This diagram illustrates the effects of queue blocking time and visualizes the probe compression phenomenon. Figure 7.8 shows two phase plots for the Internet link model (as implemented in OPNET® Modeler) with two different queue processing rates, μ. The queue processing rate roughly corresponds to the bandwidth of the bottleneck link. In keeping with [51], δ_{tx} is the transmission period for the guidance messages, and P is the packet size.

In Figure 7.8(a), the points clustered around the line $RTT(n+1) = RTT(n) + P/\mu + \delta_{tx}$ represent control packets that build up in the queue with little bandwidth between them. This is the packet compression effect described in [51]. Figure 7.8(b) shows that the compression disappears when the queue processing rate is increased. These results are consistent with the measured results presented in [51].

7.3.2.5 Background Traffic Model

The bg_sender process model is added to the vehicle node model to provide a method of injecting background traffic into the simulation.

Each vehicle's node model contains a background sender process model to provide additional traffic on the wireless network that competes with the guidance packets for available bandwidth. This process model is used in the Wireless Congestion Scenario described in Section 7.5.3. Each background sender generates packets of a fixed size at a rate determined by a Poisson random process. Each packet is addressed to another vehicle on the wireless network. The destination vehicle is chosen at random for each packet. The mean rate and the size of the packets generated can be configured through simulation parameters.



(a) Low Bandwidth, μ=50 kbps

(b) High Bandwidth, μ=200 kbps

Figure 7.8. Phase Plot for the Internet Link Model with B=exponential(20 kbps), D=100 ms, δ_{tx}=100 ms, P=120 bits

For simulations where the background sender is not used, the model can be disabled. In this case, the model does not interact with the simulation.

7.4 Simulation Configurations

This section describes the different simulation configurations that are studied in the traffic control experiments. The goal of these simulations is to compare the performance of different mechanisms in each scenario and to select scenarios that represent realistic operating conditions.

To evaluate the traffic control system, we examine several different configurations of the enterprise network. Each scenario is described in greater detail in Section 7.5. The various configurable parameters for the simulation configuration are also discussed.

7.4.1 Simulation Constants

This section describes parameters that are not varied in the simulations. They are described to assist others in reproducing this work. The OPNET® workstation models have many options, most of which are not relevant to our experiments. Any option not described here or in the following section was left at the default value. The defaults are the defaults for OPNET® Modeler version 16.0 A PL6, with the version of the model library dated September 28, 2010.

- **Wireless Link speed:** the vehicles communicate using 802.11g wireless protocols, which supports raw bitrates of 1, 2, 5, and 11 Mbps. All experiments use the lowest speed, 1 Mbps, since this reflects the most likely use of the wireless protocol in an outdoor environment.

- **Wireless Link Power:** the transmitter is configured at the default value of 0.005.

- **Routing Protocol:** all experiments use the Ad hoc On-Demand Distance Vector (AODV) routing protocol. Of the routing protocols implemented in OPNET® modeler, this one is the most fully featured. It is the only one that supports integrated routing with wired Ethernet networks. The wireless portion of the simulation was tested with other routing protocols, but the impact on the inter-arrival time of the guidance packets was negligible.

- **CA resolution:** the CA model is updated at a frequency of ten ticks/s.

- **Road Length:** the length of the ring road L is 30,000 cells or 750 m, which results in each cell being 0.025 m in length.

- **Vehicle Model Parameters:** the vehicles in the simulation are all the same size, l. They are 120 cells (3 meters) in length.

  - **Maximum velocity:** the vehicle models have a maximum velocity v_{max} of 60 cells/tick (15 m/s).
  - **Maximum acceleration:** the vehicles' models have a maximum acceleration a_{max} of 1 cell/tick/tick (2.5 m/s/s).
  - **Maximum deceleration:** the maximum deceleration is effectively infinite, which is a model constraint, as the cars will always slow down to avoid passing or intersecting with the car in front of them. The maximum deceleration, when the path of the car is not obstructed (e.g. because guidance commands a reduced speed), is the same magnitude as that of the maximum acceleration.
  - **Probability of random deceleration:** the probability of random deceleration p is 0.

- **Stoplight Parameters:** the stoplight is configured to have alternating green and red periods of 30 seconds.
- **Simulation Length:** each simulation run is 60 (simulated) minutes long.

7.4.2 Simulation Parameters

The purpose of the case study is to vary different aspects of the simulation and evaluate the performance of different gateway mechanisms in these conditions. This section describes the parameters that are varied to study the performance of mechanisms in different conditions.

7.4.2.1 Random Seed

The **Random Seed** is an OPNET® parameter that affects the random behavior of the simulator for aspects of the simulation that are governed by random distributions, including

aspects of the implementation of the network models provided by OPNET® as well as items specific to this case study, such as the random selection of packet drops for queues using the Drop Random policy. Multiple executions of the same scenario with different random seeds are used to increase confidence in the results.

7.4.2.2 Car Count

The **Car Count** is the parameter *N* in the CA model discussed in Section 7.1.3. The greater the number of vehicles on the road, the greater the load on the network. Also, the road is more congested, resulting in lower overall flow rates.

7.4.2.3 Enterprise Network Configuration

In the **Local MANET Guidance** scenario, the controller computing velocity guidance resides at the stoplight node and transmits guidance packets to vehicle nodes via a Mobile Ad-hoc Network (MANET). In the **Central MANET Guidance** scenario, the controller computing velocity guidance resides at a remote node that transmits guidance packets to the stoplight node, which then transmits them to vehicle nodes via a MANET.

In the Local MANET Guidance scenario, the stoplight node participates in an ad-hoc network with the vehicle nodes. Communication occurs via IEEE 802.11 wireless protocols with the AODV routing protocol. Guidance computations are done locally at the stoplight and distributed to the nodes over the wireless network. The network architecture is shown in Figure 7.9. Guidance information is computed locally in the stoplight node and then propagated to the vehicle nodes over the MANET.

In the Central MANET Guidance configuration, the stoplight node participates with the vehicle nodes in the same wireless network as the Local MANET Guidance configuration, but also communicates with a central server that performs the guidance calculation and returns the results to the stoplight node to distribute to the ad-hoc wireless network. This

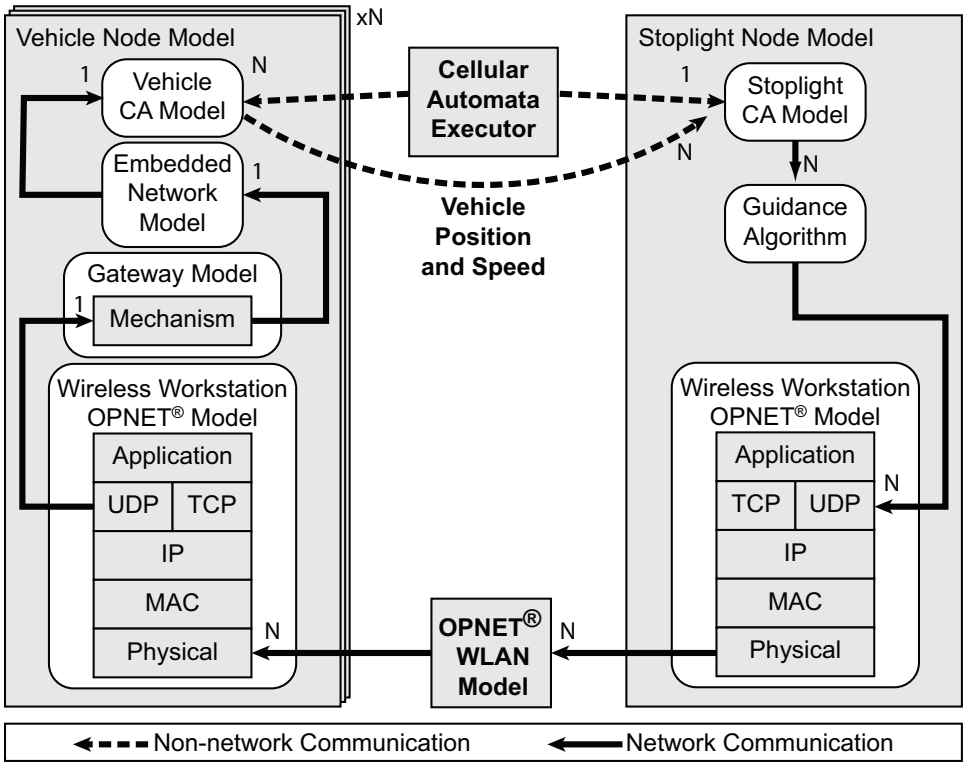


Figure 7.9. Network Architecture for Local MANET Guidance

configuration reflects the economics of deploying a traffic control system over a wide area. Rather than distribute high-cost, intelligent nodes into every stoplight, a central coordinating server performs the guidance calculations, and the stoplight nodes are simply used for local communication. This configuration provides several other advantages as well. Central servers are easier to manage, update, and replicate. Using central servers also provides the potential for non-local coordination, such as coordinated timing between multiple intersections or adjusting guidance due to perturbing events such as accidents. The network architecture for the Central MANET Guidance configuration is shown in Figure 7.10. It is similar to the Local MANET Guidance configuration, except that an additional network round trip is required to bring the guidance information from the central controller to the stoplight wireless model.

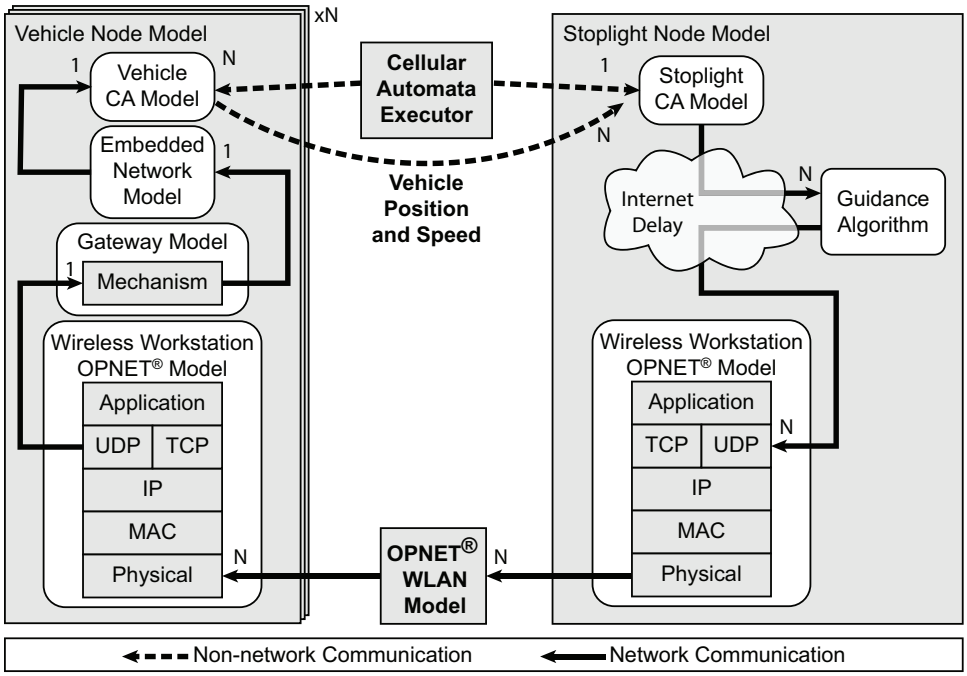


Figure 7.10. Network Architecture for Central MANET Guidance

In the Central MANET Guidance configuration, there are two additional parameters related to the model of the wired link from [51]. This model is described fully in Section 7.3.2.4. It has three parameters: processing rate μ, delay D, and Internet bitrate B. In our simulations, the processing rate μ is fixed at 138 kbps and the delay D is fixed at 0.1s (the values given in [51]). The Internet bitrate B is allowed to vary.

7.4.2.4 Packet Reception Threshold

The **Packet Reception Threshold** is a parameter of the wireless models provided by OPNET®. Raising the threshold increases the signal strength required for the simulator to deem a packet received by a particular node, effectively adding noise to the system. A special simulation with two wireless nodes is conducted to demonstrate the effect of varying this parameter on the reception distance. The results are shown in Figure 7.11.

Since the diameter of the ring road is 238.7 meters, a Packet Reception Threshold value of -85 dB (maximum reception distance of 393.5 meters) allows any node to communicate

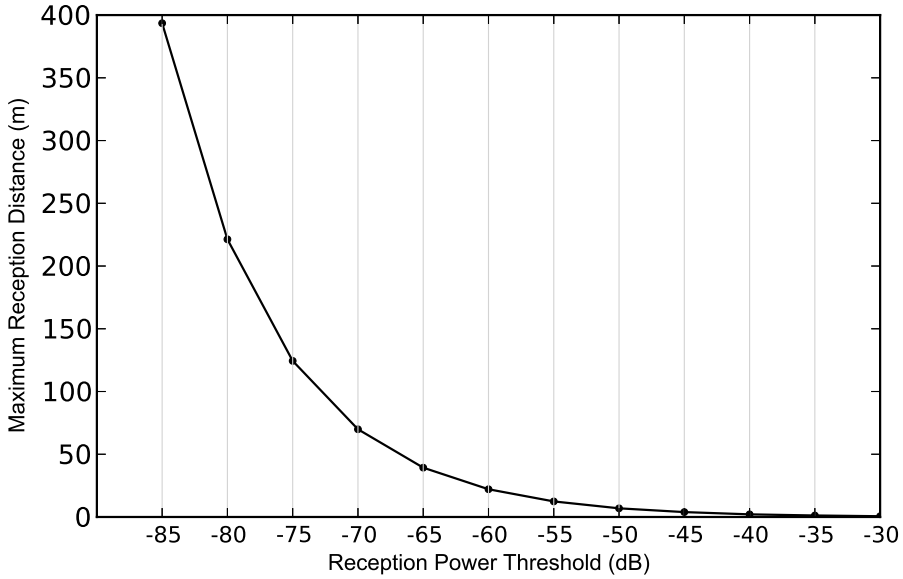


Figure 7.11. Reception Distance vs. Packet Reception Threshold

with any other node. As the threshold decreases, the reception distance decreases. The lowest feasible value of the threshold is -50 dB. Below -50 dB, the maximum reception distance is less than the minimum spacing of the vehicles, so the vehicles cannot even communicate with a neighbor that is immediately adjacent.

7.4.2.5 Guidance Update Frequency

The **Guidance Update Frequency**, the frequency at which guidance updates are generated, can be modified with a model parameter. Whenever enough time elapses to send a new guidance update, the CA guidance model generates a guidance packet for each car. Since the transmission of packets occurs in Step 5 of the CA model execution, packets are always queued for sending immediately after the CA update occurs. The embedded system is always configured to handle guidance messages at the same rate that they are sent from the server.

7.4.2.6 Background Traffic Model

The background traffic model sends additional messages between the wireless vehicle nodes in the simulation. This model is described in Section 7.3.2.5. The vehicle nodes include a process model which sends additional background packets. There are two parameters that control this model: **Background Mean Send Rate** is the mean rate for the Poisson process that sends background traffic, and the **Background Packet Size** parameter determines the size (in bytes) of the packets sent.

7.4.3 Mechanisms and Policies

The simulations are run using the following mechanisms and policies:

- **Queue Mechanisms:** The simulations include runs with queue length policies of 1, 10, 20, and 50. For bounded queues, we use the Drop Oldest overflow policy. For underflow policy, we use the mailbox policy.
- **Filter Mechanisms:** The simulations include runs with filters using the following underflow policies: Constant Extrapolation, Linear Extrapolation, and Decaying Linear Extrapolation. The Decaying Linear Extrapolation policy is used with threshold parameters of 4 and 8 seconds.

7.5 Simulation Experiments

This section describes the experiments that are run using the OPNET® simulations described above. The results are broken down by scenario. Each set of experiments is described, and the parameters which are varied during the experiments are given.

Table 7.2. Experiments for the Internet Link Congestion Scenario

Guidance Update Period	Packet RX Power Threshold	Bgnd. Mean Send Period	Bgnd. Packet Size	Mean Internet Bit Rate	Car Count	Number of Experiments
10	-85	n/a	n/a	13800	15	100
10	-85	n/a	n/a	27600	15	100
10	-85	n/a	n/a	69000	15	100
10	-85	n/a	n/a	110400	15	100
10	-85	n/a	n/a	13800	30	100
10	-85	n/a	n/a	27600	30	100
10	-85	n/a	n/a	69000	30	100
10	-85	n/a	n/a	110400	30	100

7.5.1 Internet Link Congestion Scenario

These experiments compare the performance of mechanisms in the Central MANET guidance network configuration while varying the parameters of the Internet link model. This scenario simulates real-world interference from other traffic sources affecting the link between the guidance computer and the wireless transmitter at the stoplight. Table 7.2 shows the parameters varied in this experiment.

7.5.2 Noisy Wireless Network Scenario

These experiments study the Local MANET Guidance network configuration while varying the receiver power threshold. This scenario models the effect of interference in the local wireless network. The values of the receiver power threshold studied are shown in Table 7.3.

Table 7.3. Experiments for the Noisy Wireless Network Scenario

Guidance Update Period	Packet RX Power Threshold	Bgnd. Mean Send Period	Bgnd. Packet Size	Mean Internet Bit Rate	Car Count	Number of Experiments
10	-85	n/a	n/a	n/a	15	100
10	-75	n/a	n/a	n/a	15	100
10	-65	n/a	n/a	n/a	15	100
10	-55	n/a	n/a	n/a	15	100
10	-85	n/a	n/a	n/a	30	100
10	-75	n/a	n/a	n/a	30	100
10	-65	n/a	n/a	n/a	30	100
10	-55	n/a	n/a	n/a	30	100

Table 7.4. Experiments for the Background Wireless Traffic Scenario

Guidance Update Period	Packet RX Power Threshold	Bgnd. Mean Send Period	Bgnd. Packet Size	Mean Internet Bit Rate	Car Count	Number of Experiments
10	-85	0.01	1024	n/a	15	30
10	-85	0.10	1024	n/a	15	30
10	-85	0.01	4096	n/a	15	30
10	-85	0.10	4096	n/a	15	30
10	-85	0.01	1024	n/a	30	10
10	-85	0.10	1024	n/a	30	10
10	-85	0.01	4096	n/a	30	10
10	-85	0.10	4096	n/a	30	10

7.5.3 Background Wireless Traffic Congestion Scenario

These experiments vary the background traffic load during the simulation in the Local MANET Guidance network configuration. The mechanism for injecting traffic into the simulation is described in Section 7.3.2.5. These experiments model traffic between the vehicle nodes that compete for bandwidth with the guidance packets from the stoplight. The values tested for packet size and mean send rate are shown in Table 7.4.

Table 7.5. Experiments for the Slow Update Frequency Scenario

Guidance Update Period	Packet RX Power Threshold	Bgnd. Mean Send Period	Bgnd. Packet Size	Mean Internet Bit Rate	Car Count	Number of Experiments
20	-85	n/a	n/a	n/a	15	100
50	-85	n/a	n/a	n/a	15	100
100	-85	n/a	n/a	n/a	15	100
200	-85	n/a	n/a	n/a	15	100
500	-85	n/a	n/a	n/a	15	100
20	-85	n/a	n/a	n/a	30	100
50	-85	n/a	n/a	n/a	30	100
100	-85	n/a	n/a	n/a	30	100
200	-85	n/a	n/a	n/a	30	100
500	-85	n/a	n/a	n/a	30	100

7.5.4 Slow Update Frequency Scenario

Rather than modify a parameter of the environment or network simulation, these experiments modify the frequency at which the guidance computer issues updates to the vehicle nodes. This scenario evaluates mechanisms in situations where slower guidance updates may be chosen (for example, because of constrained bandwidth). The update frequencies tested are shown in Table 7.5.

7.6 Simulation Results

This section discusses a selection of results from the traffic simulations. Results are computed by taking the mean of each metric over all the simulation runs for a particular scenario and gateway mechanism so the only parameter that varies is the random seed. The first 120 seconds of each simulation run are excluded from the results. This warm-up time allows the vehicles to get up to speed, since the simulation begins with all the cars stopped at the light. In 99% of the simulations, all of the vehicles have begun moving after 120s. In

addition to results for each metric, a 95% confidence interval is obtained using bootstrap analysis [53].

A selection of results is discussed below. The complete results for these experiments are given in Appendix B.

The first result to consider is from the Internet Link Congestion Scenario, with an Internet Mean Bit Rate of 13.8 kbps and 30 vehicles in the simulation. The results for each mechanism and metric are shown in Table 7.6 and graphed in Figure 7.12. These results show that the performance of the flow and motionless ratio metrics is the same (within the 95% confidence interval). The mean inter-arrival time of packets at the gateway for this experiment is 1.017 seconds. In this case, the network is fast enough that little timing mitigation is needed, so the queue mechanisms do better because they avoid the estimation errors that occur in the application-aware data models. Even though the flow and motionless ratio are essentially unaffected by the choice of mechanism, choosing one of the queue mechanisms or the constant-order extrapolation filter saves 0.411 L of fuel over the course of a one-hour simulation with 30 vehicles.

The next result to consider is from the Background Traffic Congestion Scenario, with a packet size of 1024 bytes, a mean period of 0.1 seconds, and 15 vehicles in the simulation. The results for each mechanism and metric are shown in Table 7.7 and graphed in Figure 7.13. Although the flows are similar for all mechanisms, the motionless ratio is reduced by a factor of 10 when the Decaying Linear Extrapolation is used. Note that the Energy-based Fuel Consumption and the Normalized Fuel Rate metrics disagree for the two DLE filters — the EFC is lower for the Filter(DLE,8), and the NFR is lower for the Filter(DLE, 4). In this case, the fuel consumption metrics do not identify a single “best” mechanism.

The results from the Internet Link Congestion Scenario with an Internet Mean Bit Rate of 13.8 kbps and 15 vehicles in the simulation show a similar pattern. The results for each

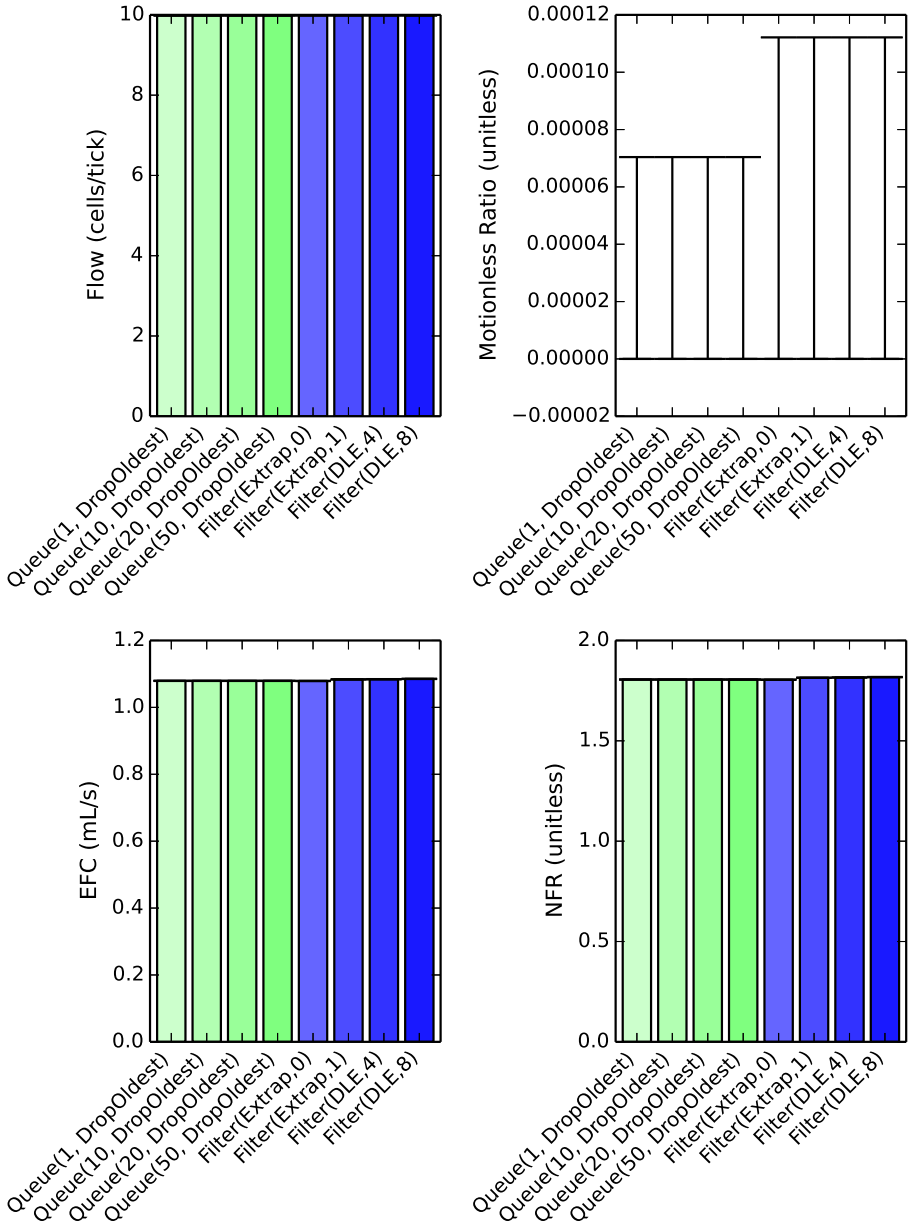


Figure 7.12. Mean Results for Internet Link Congestion Scenario with Mean Internet Bit Rate=13800 bps; Car Count=30

mechanism and metric are shown in Table 7.8 and graphed in Figure 7.14. The flow is slightly better for the filter mechanisms (vs. the queue mechanisms). In this case, the two

Table 7.6. Mean Results for Internet Link Congestion Scenario with Mean Internet Bit Rate=13800 bps; Car Count=30

Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
Queue(1, DropOldest)	**9.985** ***+1.3e-03*** ***-1.3e-03***	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.0793 *+3.7e-04* *-3.6e-04*	1.8055 *+6.0e-04* *-6.6e-04*
Queue(10, DropOldest)	**9.985** ***+1.3e-03*** ***-1.3e-03***	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.0793 *+3.7e-04* *-3.6e-04*	1.8055 *+6.0e-04* *-6.6e-04*
Queue(20, DropOldest)	**9.985** ***+1.3e-03*** ***-1.3e-03***	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.0793 *+3.7e-04* *-3.6e-04*	1.8055 *+6.0e-04* *-6.6e-04*
Queue(50, DropOldest)	**9.985** ***+1.3e-03*** ***-1.3e-03***	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.0793 *+3.7e-04* *-3.6e-04*	1.8055 *+6.0e-04* *-6.6e-04*
Filter(Extrap,0)	**9.987** ***+1.5e-03*** ***-1.5e-03***	**0.00000** ***+1.1e-04*** ***-0.0e+00***	1.0790 *+3.9e-04* *-3.8e-04*	1.8045 *+6.6e-04* *-6.5e-04*
Filter(Extrap,1)	**9.987** ***+1.5e-03*** ***-1.5e-03***	**0.00000** ***+1.1e-04*** ***-0.0e+00***	1.0836 *+3.9e-04* *-4.0e-04*	1.8143 *+6.5e-04* *-7.1e-04*
Filter(DLE,4)	**9.987** ***+1.5e-03*** ***-1.5e-03***	**0.00000** ***+1.1e-04*** ***-0.0e+00***	1.0842 *+3.8e-04* *-4.1e-04*	1.8156 *+6.4e-04* *-6.9e-04*
Filter(DLE,8)	**9.987** ***+1.5e-03*** ***-1.5e-03***	**0.00000** ***+1.1e-04*** ***-0.0e+00***	1.0852 *+3.8e-04* *-4.0e-04*	1.8175 *+6.3e-04* *-7.1e-04*

fuel consumption metrics agree, but the fuel consumtpion results among the filter mechanisms are similar enough (within the 95% confidence interval) that the fuel consumption cannot be used to distinguish between them.

The next result is shown in Figure 7.15 and Table 7.9. It is a result from the Wireless Traffic Congestion Scenario. In this experiment, background traffic with a packet size of 4096 bytes is produced by a Poisson process with a mean inter-arrival period of 0.01 seconds. There are 15 cars in the experiment. Based on the flow metric (higher is better) and

Table 7.7. Mean Results for Background Wireless Traffic Scenario with Bgnd. Packet Size=1024 bytes; Bgnd. Mean Send Period=0.10 s; Car Count=15

Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
Queue(1, DropOldest)	**4.996** ***+5.2e-03*** ***-5.6e-03***	0.00366 *+2.5e-04* *-2.4e-04*	1.1579 *+1.3e-03* *-1.2e-03*	2.0297 *+2.2e-03* *-2.2e-03*
Queue(10, DropOldest)	**4.996** ***+5.2e-03*** ***-5.6e-03***	0.00366 *+2.5e-04* *-2.4e-04*	1.1579 *+1.3e-03* *-1.2e-03*	2.0297 *+2.2e-03* *-2.2e-03*
Queue(20, DropOldest)	**4.996** ***+5.2e-03*** ***-5.6e-03***	0.00366 *+2.5e-04* *-2.4e-04*	1.1579 *+1.3e-03* *-1.2e-03*	2.0297 *+2.2e-03* *-2.2e-03*
Queue(50, DropOldest)	**4.996** ***+5.2e-03*** ***-5.6e-03***	0.00366 *+2.5e-04* *-2.4e-04*	1.1579 *+1.3e-03* *-1.2e-03*	2.0297 *+2.2e-03* *-2.2e-03*
Filter(Extrap,0)	**4.998** ***+4.8e-03*** ***-5.1e-03***	0.00344 *+2.6e-04* *-2.5e-04*	1.1585 *+1.1e-03* *-1.2e-03*	2.0261 *+2.0e-03* *-2.2e-03*
Filter(Extrap,1)	**4.993** ***+6.6e-03*** ***-6.3e-03***	0.01535 *+6.1e-04* *-5.8e-04*	1.3445 *+1.6e-03* *-1.6e-03*	2.4249 *+3.0e-03* *-3.0e-03*
Filter(DLE,4)	**4.997** ***+4.5e-03*** ***-4.3e-03***	**0.00023** ***+2.0e-04*** ***-1.8e-04***	1.3874 *+1.7e-03* *-1.7e-03*	2.4274 *+2.9e-03* *-3.0e-03*
Filter(DLE,8)	**4.996** ***+6.1e-03*** ***-5.6e-03***	**0.00020** ***+2.1e-04*** ***-1.9e-04***	1.3733 *+1.5e-03* *-1.5e-03*	2.4441 *+2.7e-03* *-2.8e-03*

the motionless ratio metric (lower is better), the constant-order extrapolating filter unambiguously outperforms the other mechanisms. However, the constant-order extrapolating filter is *worst* among the mechanisms when it comes to either of the fuel consumption metrics. The reason is that more fuel is consumed while driving than while idling. In the next best result (any of the queue mechanism simulations), the vehicles spend twice as much time at rest as they do in the simulations using the constant-order extrapolation filter.

Table 7.8. Mean Results for Internet Link Congestion Scenario with Mean Internet Bit Rate=13800 bps; Car Count=15

Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
Queue(1, DropOldest)	4.997 *+5.3e-04* *-5.0e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.0830 *+5.2e-04* *-5.6e-04*	1.8104 *+8.9e-04* *-9.4e-04*
Queue(10, DropOldest)	4.997 *+5.3e-04* *-5.0e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.0830 *+5.2e-04* *-5.6e-04*	1.8104 *+8.9e-04* *-9.4e-04*
Queue(20, DropOldest)	4.997 *+5.3e-04* *-5.0e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.0830 *+5.2e-04* *-5.6e-04*	1.8104 *+8.9e-04* *-9.4e-04*
Queue(50, DropOldest)	4.997 *+5.3e-04* *-5.0e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.0830 *+5.2e-04* *-5.6e-04*	1.8104 *+8.9e-04* *-9.4e-04*
Filter(Extrap,0)	**4.999** ***+6.1e-04*** ***-6.6e-04***	**0.00000** ***+1.0e-04*** ***-0.0e+00***	1.0814 *+5.8e-04* *-5.3e-04*	1.8075 *+9.3e-04* *-9.1e-04*
Filter(Extrap,1)	**4.999** ***+6.2e-04*** ***-6.8e-04***	**0.00000** ***+1.0e-04*** ***-0.0e+00***	1.0820 *+5.5e-04* *-5.3e-04*	1.8087 *+9.3e-04* *-9.0e-04*
Filter(DLE,4)	**4.999** ***+6.1e-04*** ***-6.8e-04***	**0.00000** ***+1.0e-04*** ***-0.0e+00***	1.0820 *+5.5e-04* *-5.3e-04*	1.8087 *+9.2e-04* *-9.0e-04*
Filter(DLE,8)	**4.999** ***+6.3e-04*** ***-6.8e-04***	**0.00000** ***+1.0e-04*** ***-0.0e+00***	1.0820 *+5.6e-04* *-5.5e-04*	1.8087 *+9.1e-04* *-9.4e-04*

These selected results give a few important insights into how metrics should be used to compare mechanism performance. In Chapter 8, we extend these ideas further by developing selection criteria based on the simulation results.

7.7 Summary

In this chapter, we use a case study to evaluate gateway mechanisms and policies and show that application-aware mechanisms can improve performance in a realistic application.

Table 7.9. Mean Results for Background Wireless Traffic Scenario with Bgnd. Packet Size=4096 bytes; Bgnd. Mean Send Period=0.01 s; Car Count=15

Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
Queue(1, DropOldest)	4.792 *+8.1e-03* *-7.1e-03*	0.01980 *+6.7e-04* *-6.6e-04*	1.0474 *+7.6e-04* *-8.4e-04*	1.8535 *+1.6e-03* *-1.7e-03*
Queue(10, DropOldest)	4.792 *+8.1e-03* *-7.1e-03*	0.01980 *+6.7e-04* *-6.6e-04*	1.0474 *+7.6e-04* *-8.4e-04*	1.8535 *+1.6e-03* *-1.7e-03*
Queue(20, DropOldest)	4.792 *+8.1e-03* *-7.1e-03*	0.01980 *+6.7e-04* *-6.6e-04*	1.0474 *+7.6e-04* *-8.4e-04*	1.8535 *+1.6e-03* *-1.7e-03*
Queue(50, DropOldest)	4.792 *+8.1e-03* *-7.1e-03*	0.01980 *+6.7e-04* *-6.6e-04*	1.0474 *+7.6e-04* *-8.4e-04*	1.8535 *+1.6e-03* *-1.7e-03*
Filter(Extrap,0)	**4.951** ***+5.4e-03*** ***-6.4e-03***	**0.00939** ***+5.2e-04*** ***-4.9e-04***	1.0708 *+9.6e-04* *-8.6e-04*	1.8857 *+2.0e-03* *-1.9e-03*
Filter(Extrap,1)	3.159 *+1.3e-02* *-1.4e-02*	0.31272 *+2.4e-03* *-2.4e-03*	0.8986 *+9.2e-04* *-1.0e-03*	1.7319 *+2.1e-03* *-2.3e-03*
Filter(DLE,4)	3.784 *+1.2e-02* *-1.2e-02*	0.19567 *+2.0e-03* *-1.8e-03*	0.9544 *+9.2e-04* *-8.9e-04*	1.7410 *+1.8e-03* *-1.8e-03*
Filter(DLE,8)	2.883 *+1.3e-02* *-1.4e-02*	0.35190 *+2.7e-03* *-2.7e-03*	0.8506 *+9.2e-04* *-8.9e-04*	1.6450 *+2.1e-03* *-1.6e-03*

The case study is a traffic control application described in [8]. The application uses a guidance algorithm to control the speed of vehicles traveling around a ring road so that they arrive at a single stoplight when it is green. This algorithm is fully described in Section 7.1.

One reason that the case study results are useful is that they use application-specific metrics (rather than the error metrics used in the abstract network simulations). Section 7.2 describes four metrics. Flow and motionless ratio are from the original application

description [8]. Two more fuel efficiency metrics from the literature [47, 48] are also used. Section 7.2 describes the metrics in greater detail.

In order to use the guidance application and simulate the behavior of wired and wireless networks, the application is implemented in the OPNET® Modeler simulation environment. The implementation combines the network simulation models provided by OPNET® with the cellular automata models of vehicle flow. Section 7.3 describes the details of the implementation.

Section 7.5 describes the four different simulation scenarios that are used. Each configuration represents a realistic scenario where disturbances in the enterprise network might affect the arrival of information at the gateway. One scenario includes an internet link that acts as a bottleneck for guidance packets. The second scenario models a noisy wireless environment. The third scenario contains background traffic that competes for network resources. The fourth scenario varies the update frequency of the application.

In these scenarios, the performance of generic (queue) and application-aware (filter) mechanisms are compared using several different configuration policies for each mechanism. These results show that, in some cases, the performance of the all the mechanisms is essentially the same. However, in a majority of the cases, the filter mechanisms offer improvement over the queue mechanisms. A selection of these results is presented in Section 7.6, and the full results are reproduced in Appendix B.

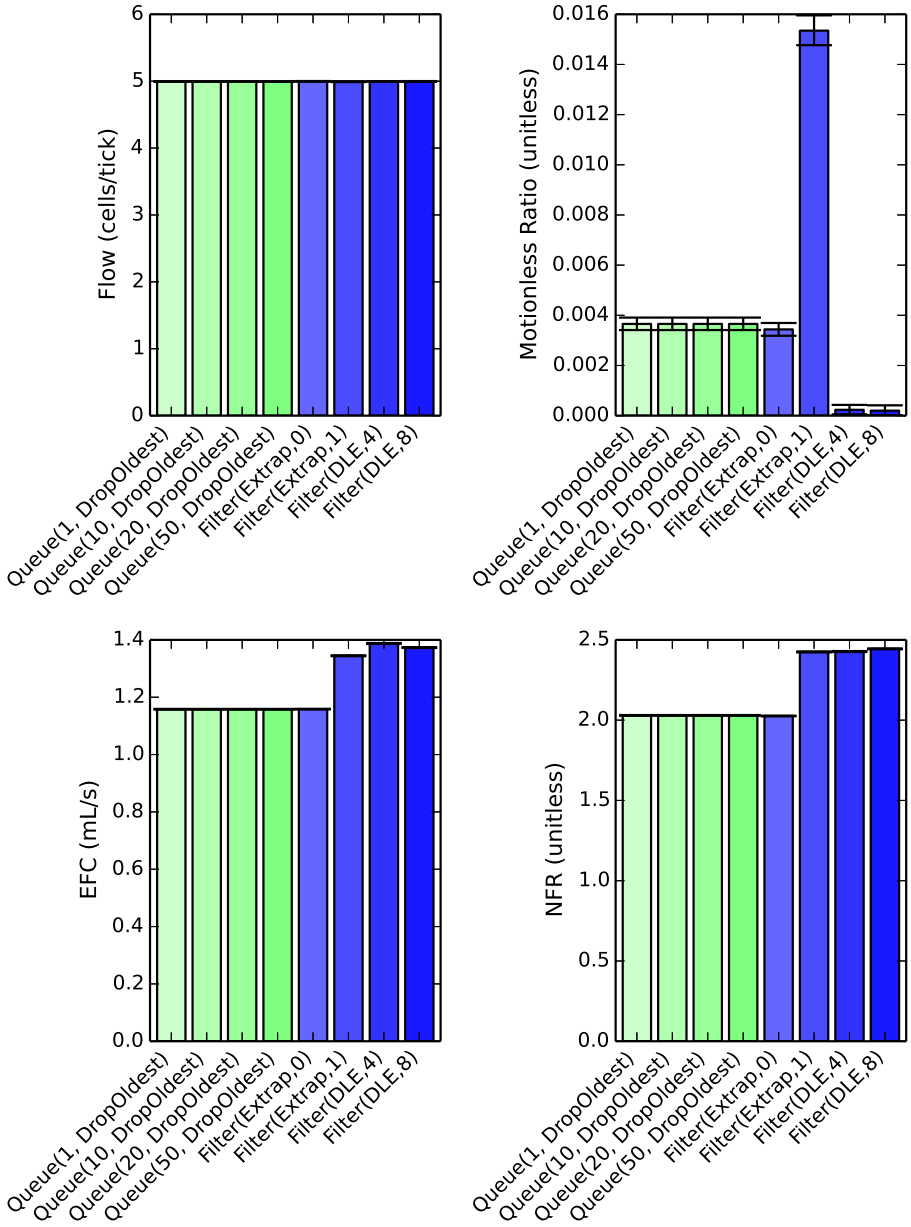


Figure 7.13. Mean Results for Background Wireless Traffic Scenario with Bgnd. Packet Size=1024 bytes; Bgnd. Mean Send Period=0.10 s; Car Count=15

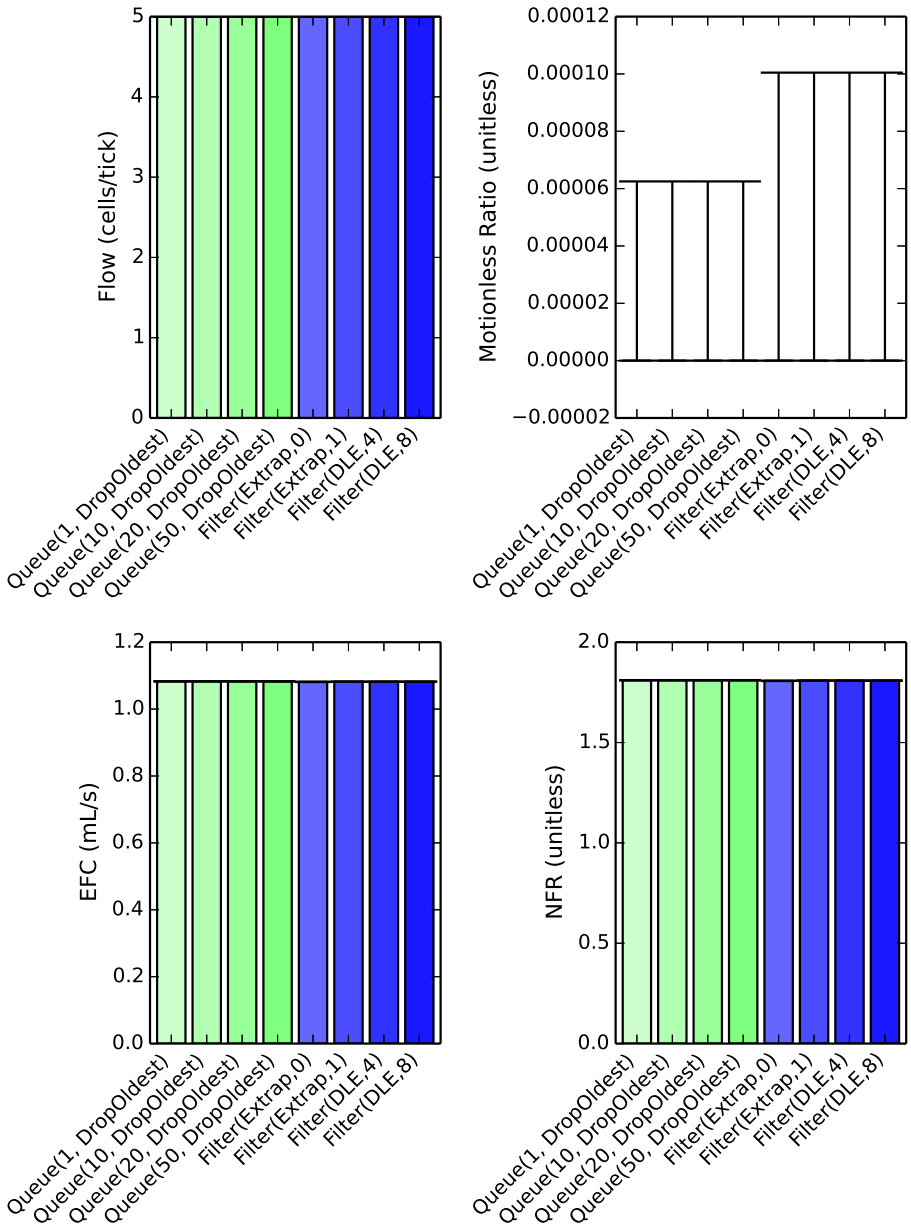


Figure 7.14. Mean Results for Internet Link Congestion Scenario with Mean Internet Bit Rate=13800 bps; Car Count=15

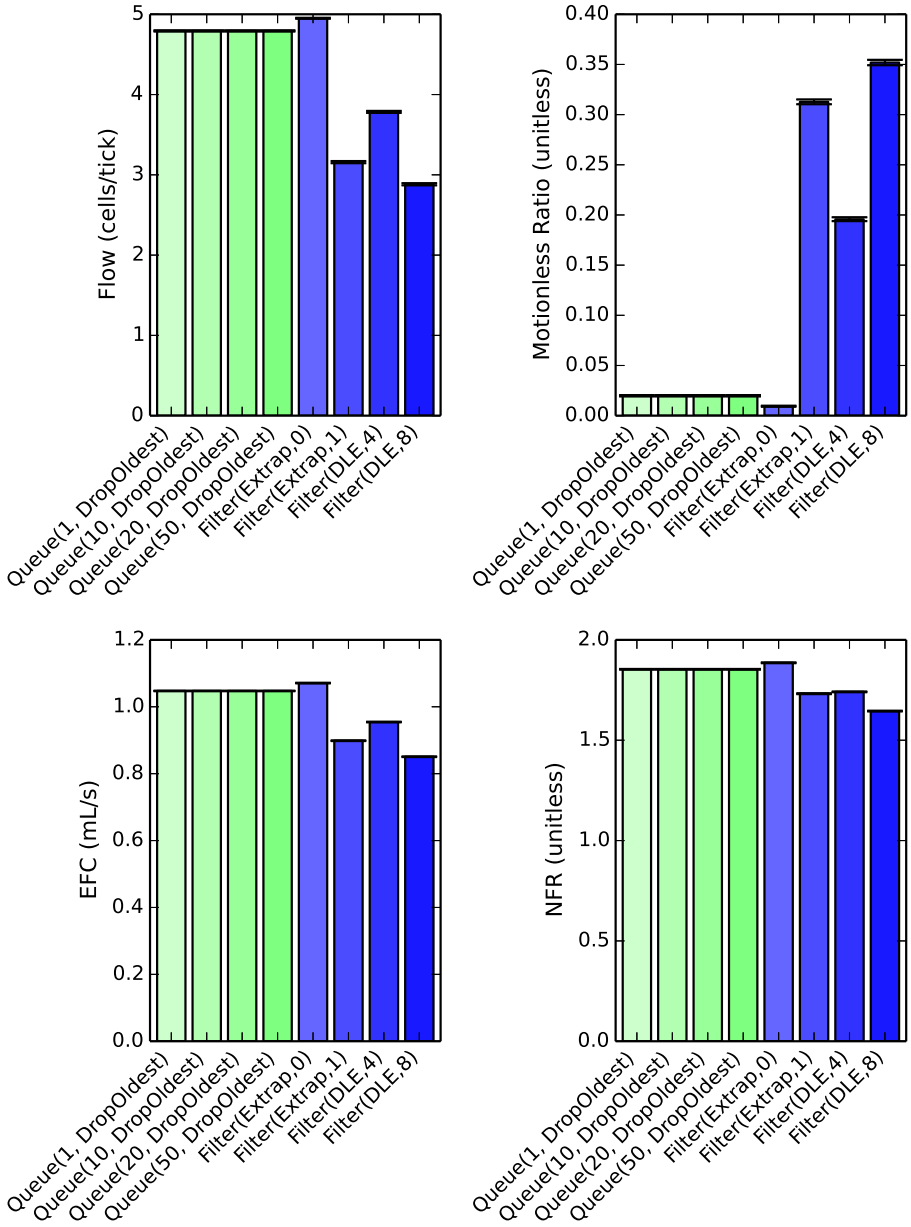


Figure 7.15. Mean Results for Background Wireless Traffic Scenario with Bgnd. Packet Size=4096 bytes; Bgnd. Mean Send Period=0.01 s; Car Count=15

Chapter 8

Gateway Mechanism Selection

This chapter describes selection rules that use the metrics from the traffic case study (see Section 7.2) to select the best mechanism for each scenario we studied (see Section 7.5). By comparing the outcome of the selection rules from each scenario to the network characteristics of that scenario, we are able to gain some insights about the relationship between the "good" mechanisms and the network characteristics for this particular application.

8.1 Basic Selection Rule

A selection rule is a rule for choosing which mechanism to use based on the metrics obtained from the traffic simulation case study. Recall that there are four metrics to consider: flow, motionless ratio, energy-based fuel consumption, and normalized fuel rate.

First, we consider a selection rule based only on flow and motionless ratio (the two metrics defined with the traffic control algorithm in [8]).

Define F as the set of mechanisms whose flow is the highest within the 95% confidence interval bound. F may contain more than one mechanism if one or more mechanisms' confidence interval overlaps the confidence interval of the mechanism with the highest flow value for the experiment. Similarly, define M as the set of mechanisms whose motionless ratios are lowest within the 95% confidence interval. Using these definitions, the selection rule is:

1. If $F \cap M$ is not empty, select $F \cap M$.

2. Otherwise, select F.

This rule prioritizes the flow metric over the motionless ratio metric, since the motionless ratios are the same in many cases, especially for scenarios where there is relatively little delay. Applying this rule to the metrics measured in the traffic simulations yields the results given in Table 8.1. These results show that filter mechanisms usually perform better than queue mechanisms, and should be preferred in most cases. However, these results do not give much insight about which filter mechanism to use. In most cases, the selection rule does not distinguish among them. For this reason, the next section proposes a second selection rule that takes fuel consumption metrics into account.

8.2 Fuel Consumption Selection Rule

In order to further refine the selection rule, we propose a modification that includes the fuel consumption metrics. Based on the observations in Section 7.6, the fuel consumption metrics are not suitable for choosing a mechanism unless the flow and motionless ratio are already similar. Thus, the fuel consumption metrics are used as a tie breaker among the mechanisms selected by the flow and motionless ratio metrics. The fuel consumption metrics do not always agree on which mechanism is better, which is the motivation behind part 3.d. of the selection rule below.

As before, define F as the set of mechanisms whose flow is the highest within the 95% confidence interval bound and M as the set of mechanisms whose motionless ratios are lowest within the 95% confidence interval. Using these definitions, the selection rule is:

1. If $F \cap M$ is empty, select F.

Table 8.1. Basic Selection Rule Results for All Experiments

Car Count	Guidance Update Period	Mean Internet Bit Rate	Packet RX Power Threshold	Bgnd. Packet Size	Bgnd. Mean Send Period	Queue(1, DropOldest)	Queue(10, DropOldest)	Queue(20, DropOldest)	Queue(50, DropOldest)	Filter(Extrap,0)	Filter(Extrap,1)	Filter(DLE,4)	Filter(DLE,8)
15	n/a	n/a	n/a	1024	0.01	X	X	X	X				
15	n/a	n/a	n/a	1024	0.10							X	X
15	n/a	n/a	n/a	4096	0.01					X			
15	n/a	n/a	n/a	4096	0.10							X	
30	n/a	n/a	n/a	1024	0.01					X			
30	n/a	n/a	n/a	1024	0.10					X			
30	n/a	n/a	n/a	4096	0.01					X			
30	n/a	n/a	n/a	4096	0.10					X			
15	n/a	13800	n/a	n/a	n/a					X	X	X	X
15	n/a	27600	n/a	n/a	n/a					X	X	X	X
15	n/a	69000	n/a	n/a	n/a					X	X	X	X
15	n/a	110400	n/a	n/a	n/a					X	X	X	X
30	n/a	13800	n/a	n/a	n/a	X	X	X	X	X	X	X	X
30	n/a	27600	n/a	n/a	n/a	X	X	X	X	X	X	X	X
30	n/a	69000	n/a	n/a	n/a					X	X	X	X
30	n/a	110400	n/a	n/a	n/a					X		X	X
15	n/a	n/a	-85	n/a	n/a					X	X	X	X
15	n/a	n/a	-75	n/a	n/a					X		X	X
15	n/a	n/a	-65	n/a	n/a					X			
15	n/a	n/a	-55	n/a	n/a					X			
30	n/a	n/a	-85	n/a	n/a	X	X	X	X	X	X	X	X
30	n/a	n/a	-75	n/a	n/a	X	X	X	X	X		X	X
30	n/a	n/a	-65	n/a	n/a					X			
30	n/a	n/a	-55	n/a	n/a	X	X	X	X				
15	20	n/a	n/a	n/a	n/a					X	X	X	X
15	50	n/a	n/a	n/a	n/a					X			
15	100	n/a	n/a	n/a	n/a	X	X	X	X				
15	200	n/a	n/a	n/a	n/a	X	X	X	X				
15	500	n/a	n/a	n/a	n/a	X	X	X	X				
30	20	n/a	n/a	n/a	n/a					X	X	X	X
30	50	n/a	n/a	n/a	n/a					X			
30	100	n/a	n/a	n/a	n/a	X	X	X	X				
30	200	n/a	n/a	n/a	n/a	X	X	X	X				
30	500	n/a	n/a	n/a	n/a	X	X	X	X				

2. If $F \cap M$ has size one, select $F \cap M$.

3. If $F \cap M$ has size greater than one, break the tie with fuel consumption metrics.

   (a) Compute C_e, the set of mechanisms from $F \cap M$ with the lowest energy-based fuel consumption within the 95% confidence interval.

   (b) Compute C_n, the set of mechanisms from $F \cap M$ with the lowest normalized fuel rate within the 95% confidence interval.

   (c) If $C_e \cap C_n$ is not empty, select $C_e \cap C_n$.

   (d) Otherwise, select $C_e \cup C_n$

Applying the selection rule defined in Section 8.1, we obtain the results shown in Table 8.1. As before, filter mechanisms usually provide improvement over queue mechanisms and should be preferred in most cases. However, adding fuel consumption helps distinguish between the filter mechanisms. The selection rule results demonstrate that in only a few cases did any other filter exceed the performance of the constant-order extrapolating filter.

8.3 A Rule for Mechanism Selection

In order to understand how the characteristics of the network affect mechanism selection, results from 8.2 are augmented with the mean inter-arrival time of guidance packets at the gateway for that experiment. Then the results are sorted by mean inter-arrival time. These results are shown in Tables 8.3 and 8.4.

Based on these results, the general rule for mechanism selection is: a filter with a constant-order extrapolating underflow policy model should be used by default for most systems. If the mean inter-arrival time of the enterprise network is less than four times the control period of the system, then other underflow policies for filter mechanisms should be

Table 8.2. Fuel Consumption Selection Rule Results for All Experiments

Car Count	Guidance Update Period	Mean Internet Bit Rate	Packet RX Power Threshold	Bgnd. Packet Size	Bgnd. Mean Send Period	Queue(1, DropOldest)	Queue(10, DropOldest)	Queue(20, DropOldest)	Queue(50, DropOldest)	Filter(Extrap,0)	Filter(Extrap,1)	Filter(DLE,4)	Filter(DLE,8)
15	n/a	n/a	n/a	1024	0.01	X	X	X	X				
15	n/a	n/a	n/a	1024	0.10							X	X
15	n/a	n/a	n/a	4096	0.01					X			
15	n/a	n/a	n/a	4096	0.10							X	
30	n/a	n/a	n/a	1024	0.01					X			
30	n/a	n/a	n/a	1024	0.10					X			
30	n/a	n/a	n/a	4096	0.01					X			
30	n/a	n/a	n/a	4096	0.10					X			
15	n/a	13800	n/a	n/a	n/a					X	X	X	X
15	n/a	27600	n/a	n/a	n/a					X	X		X
15	n/a	69000	n/a	n/a	n/a					X			
15	n/a	110400	n/a	n/a	n/a					X			
30	n/a	13800	n/a	n/a	n/a	X	X	X	X	X			
30	n/a	27600	n/a	n/a	n/a					X			
30	n/a	69000	n/a	n/a	n/a					X			
30	n/a	110400	n/a	n/a	n/a					X			
15	n/a	n/a	-85	n/a	n/a					X	X	X	X
15	n/a	n/a	-75	n/a	n/a					X			
15	n/a	n/a	-65	n/a	n/a					X			
15	n/a	n/a	-55	n/a	n/a					X			
30	n/a	n/a	-85	n/a	n/a					X			
30	n/a	n/a	-75	n/a	n/a	X	X	X	X				
30	n/a	n/a	-65	n/a	n/a					X			
30	n/a	n/a	-55	n/a	n/a	X	X	X	X				
15	20	n/a	n/a	n/a	n/a					X	X	X	X
15	50	n/a	n/a	n/a	n/a					X			
15	100	n/a	n/a	n/a	n/a	X	X	X	X				
15	200	n/a	n/a	n/a	n/a	X	X	X	X				
15	500	n/a	n/a	n/a	n/a	X	X	X	X				
30	20	n/a	n/a	n/a	n/a					X			
30	50	n/a	n/a	n/a	n/a					X			
30	100	n/a	n/a	n/a	n/a	X	X	X	X				
30	200	n/a	n/a	n/a	n/a	X	X	X	X				
30	500	n/a	n/a	n/a	n/a	X	X	X	X				

Table 8.3. Basic Selection Rule Results Sorted by Experiment Mean Inter-arrival Time

Car Count	Guidance Update Period	Mean Internet Bit Rate	Packet RX Power Threshold	Bgnd. Packet Size	Bgnd. Mean Send Period	Queue(1, DropOldest)	Queue(10, DropOldest)	Queue(20, DropOldest)	Queue(50, DropOldest)	Filter(Extrap,0)	Filter(Extrap,1)	Filter(DLE,4)	Filter(DLE,8)	Inter-arrival Mean
15	n/a	13800	n/a	n/a	n/a					X	X	X	X	1.001
15	n/a	27600	n/a	n/a	n/a					X	X	X	X	1.001
15	n/a	n/a	-85	n/a	n/a					X	X	X	X	1.001
15	n/a	69000	n/a	n/a	n/a					X	X	X	X	1.014
30	n/a	27600	n/a	n/a	n/a	X	X	X	X	X	X	X	X	1.017
30	n/a	13800	n/a	n/a	n/a	X	X	X	X	X	X	X	X	1.018
30	n/a	69000	n/a	n/a	n/a					X	X	X	X	1.032
15	n/a	110400	n/a	n/a	n/a					X	X	X	X	1.053
30	n/a	n/a	-85	n/a	n/a	X	X	X	X	X	X	X	X	1.062
30	n/a	110400	n/a	n/a	n/a					X		X	X	1.086
15	20	n/a	n/a	n/a	n/a					X	X	X	X	2.003
30	20	n/a	n/a	n/a	n/a					X	X	X	X	2.067
15	n/a	n/a	n/a	1024	0.10							X	X	2.559
30	n/a	n/a	-65	n/a	n/a					X				2.946
15	n/a	n/a	n/a	4096	0.10							X		4.102
30	n/a	n/a	-55	n/a	n/a	X	X	X	X					4.262
30	n/a	n/a	-75	n/a	n/a	X	X	X	X	X		X	X	4.696
15	n/a	n/a	-75	n/a	n/a					X		X	X	5.331
15	n/a	n/a	n/a	1024	0.01	X	X	X	X					6.446
15	n/a	n/a	-65	n/a	n/a					X				6.540
15	n/a	n/a	n/a	4096	0.01					X				9.523
30	50	n/a	n/a	n/a	n/a					X				10.532
15	n/a	n/a	-55	n/a	n/a					X				10.648
15	50	n/a	n/a	n/a	n/a					X				16.722
30	200	n/a	n/a	n/a	n/a	X	X	X	X					24.664
15	200	n/a	n/a	n/a	n/a	X	X	X	X					25.353
30	n/a	n/a	n/a	4096	0.10					X				26.313
30	n/a	n/a	n/a	1024	0.10					X				26.772
30	n/a	n/a	n/a	4096	0.01					X				28.882
30	n/a	n/a	n/a	1024	0.01					X				29.083
15	500	n/a	n/a	n/a	n/a	X	X	X	X					50.000
30	500	n/a	n/a	n/a	n/a	X	X	X	X					50.000
30	100	n/a	n/a	n/a	n/a	X	X	X	X					170.851
15	100	n/a	n/a	n/a	n/a	X	X	X	X					351.409

Table 8.4. Fuel Consumption Selection Rule Results Sorted by Experiment Mean Inter-arrival Time

Car Count	Guidance Update Period	Mean Internet Bit Rate	Packet RX Power Threshold	Bgnd. Packet Size	Bgnd. Mean Send Period	Queue(1, DropOldest)	Queue(10, DropOldest)	Queue(20, DropOldest)	Queue(50, DropOldest)	Filter(Extrap,0)	Filter(Extrap,1)	Filter(DLE,4)	Filter(DLE,8)	Inter-arrival Mean
15	n/a	13800	n/a	n/a	n/a					X	X	X	X	1.001
15	n/a	27600	n/a	n/a	n/a					X	X		X	1.001
15	n/a	n/a	-85	n/a	n/a					X	X	X	X	1.001
15	n/a	69000	n/a	n/a	n/a					X				1.014
30	n/a	27600	n/a	n/a	n/a					X				1.017
30	n/a	13800	n/a	n/a	n/a	X	X	X	X	X				1.018
30	n/a	69000	n/a	n/a	n/a					X				1.032
15	n/a	110400	n/a	n/a	n/a					X				1.053
30	n/a	n/a	-85	n/a	n/a					X				1.062
30	n/a	110400	n/a	n/a	n/a					X				1.086
15	20	n/a	n/a	n/a	n/a					X	X	X	X	2.003
30	20	n/a	n/a	n/a	n/a					X				2.067
15	n/a	n/a	n/a	1024	0.10							X	X	2.559
30	n/a	n/a	-65	n/a	n/a					X				2.946
15	n/a	n/a	n/a	4096	0.10							X		4.102
30	n/a	n/a	-55	n/a	n/a	X	X	X	X					4.262
30	n/a	n/a	-75	n/a	n/a	X	X	X	X					4.696
15	n/a	n/a	-75	n/a	n/a					X				5.331
15	n/a	n/a	n/a	1024	0.01	X	X	X	X					6.446
15	n/a	n/a	-65	n/a	n/a					X				6.540
15	n/a	n/a	n/a	4096	0.01					X				9.523
30	50	n/a	n/a	n/a	n/a					X				10.532
15	n/a	n/a	-55	n/a	n/a					X				10.648
15	50	n/a	n/a	n/a	n/a					X				16.722
30	200	n/a	n/a	n/a	n/a	X	X	X	X					24.664
15	200	n/a	n/a	n/a	n/a	X	X	X	X					25.353
30	n/a	n/a	n/a	4096	0.10					X				26.313
30	n/a	n/a	n/a	1024	0.10					X				26.772
30	n/a	n/a	n/a	4096	0.01					X				28.882
30	n/a	n/a	n/a	1024	0.01					X				29.083
15	500	n/a	n/a	n/a	n/a	X	X	X	X					50.000
30	500	n/a	n/a	n/a	n/a	X	X	X	X					50.000
30	100	n/a	n/a	n/a	n/a	X	X	X	X					170.851
15	100	n/a	n/a	n/a	n/a	X	X	X	X					351.409

tested through simulation. If the mean inter-arrival time is greater than 30 times the control period of the system, then a queue mechanism should be used.

8.4 Improvements Due to Mechanism Selection

This section describes the improvements gained by selecting a mechanism using the fuel consumption selection rule described in Section 8.2. Here, we define improvement as the difference in a given metric between the selected mechanism and the best unselected mechanism. Improvement is always positive, regardless of the sense of the metric (i.e. higher flow values are better, but lower motionless ratio and fuel consumption values are better). If more than one mechanism is selected by the rule (a tie), then the worst value is used to measure improvement.

The results for the improvement measurements are shown in Table 8.5. By applying the selection rules, we obtain improvements in the flow metric of up to 44.8% (with six out of 32 experiments having an improvement of greater than 35%). The fuel consumption improvements vary between a loss (negative improvement) of 18% to a positive 18% improvement. This variation is due in part to the fact that the fuel consumption metric is used to break selection ties, not as a primary selection criteria. The 200% improvement in motionless ratio is moderated by the fact that motionless ratios are, in general, small numbers, so the large percentage change is not as meaningful.

8.5 Summary

In this chapter, we use the simulation results from Chapter 7 to develop selection rules — heuristic guidelines for selecting mechanisms based on the performance of the four application metrics. Two selection rules (the second a refinement of the first) are developed.

Table 8.5. Improvements Due to Mechanism Selection

Car Count	Guidance Update Period	Mean Internet Bit Rate	Packet RX Power Threshold	Bgnd. Packet Size	Bgnd. Mean Send Period	% improvement over best unselected mech.			
						Flow	Motionless Ratio	EFC	NFR
15	n/a	n/a	n/a	1024.0	0.01	1.1	37.9	-1.1	-1.1
15	n/a	n/a	n/a	1024.0	0.1	-0.0	178.0	-17.0	-18.7
15	n/a	n/a	n/a	4096.0	0.01	3.3	71.3	-2.2	-1.7
15	n/a	n/a	n/a	4096.0	0.1	-0.2	68.0	-7.7	-7.1
30	n/a	n/a	n/a	1024.0	0.01	38.5	152.7	-7.4	-2.9
30	n/a	n/a	n/a	1024.0	0.1	44.8	-7.3	-12.2	-7.6
30	n/a	n/a	n/a	4096.0	0.01	37.0	150.7	-2.4	1.1
30	n/a	n/a	n/a	4096.0	0.1	42.0	0.8	-11.2	-6.5
15	n/a	13800.0	n/a	n/a	n/a	0.0	0.0	0.1	0.1
15	n/a	27600.0	n/a	n/a	n/a	-0.0	0.0	0.0	0.0
15	n/a	69000.0	n/a	n/a	n/a	0.0	0.0	0.5	0.6
15	n/a	110400.0	n/a	n/a	n/a	0.0	0.0	1.5	1.8
30	n/a	13800.0	n/a	n/a	n/a	-0.0	0.0	0.4	0.5
30	n/a	27600.0	n/a	n/a	n/a	0.0	0.0	0.6	0.8
30	n/a	69000.0	n/a	n/a	n/a	0.0	0.0	1.2	1.5
30	n/a	110400.0	n/a	n/a	n/a	0.0	-190.0	2.7	3.3
15	n/a	n/a	-85.0	n/a	n/a	0.0	0.0	0.0	0.0
15	n/a	n/a	-75.0	n/a	n/a	0.0	190.6	3.2	4.0
15	n/a	n/a	-65.0	n/a	n/a	0.0	9.8	0.2	0.3
15	n/a	n/a	-55.0	n/a	n/a	0.6	-2.2	-1.8	-3.0
30	n/a	n/a	-85.0	n/a	n/a	0.0	200.0	1.7	2.0
30	n/a	n/a	-75.0	n/a	n/a	-0.0	43.6	0.3	0.4
30	n/a	n/a	-65.0	n/a	n/a	0.8	66.3	-0.6	-0.2
30	n/a	n/a	-55.0	n/a	n/a	2.1	47.1	-1.0	-0.4
15	20.0	n/a	n/a	n/a	n/a	0.1	0.0	-0.1	-0.1
15	50.0	n/a	n/a	n/a	n/a	0.2	-19.9	-0.2	-0.1
15	100.0	n/a	n/a	n/a	n/a	36.0	110.5	-11.6	-7.4
15	200.0	n/a	n/a	n/a	n/a	3.3	34.6	-2.2	-1.9
15	500.0	n/a	n/a	n/a	n/a	5.3	31.4	-2.4	-1.2
30	20.0	n/a	n/a	n/a	n/a	0.0	0.0	1.7	2.2
30	50.0	n/a	n/a	n/a	n/a	0.1	143.1	15.8	18.9
30	100.0	n/a	n/a	n/a	n/a	15.7	38.3	-3.1	-2.2
30	200.0	n/a	n/a	n/a	n/a	43.9	23.6	-14.1	-8.9
30	500.0	n/a	n/a	n/a	n/a	15.5	42.6	-6.9	-3.1
Maximum improvment						44.8	200.0	15.8	18.9

The basic selection rule uses only the flow and motionless ratio metrics that are given in the original application description [8]. Applying the rule to the results shows that filter mechanisms are selected more often than queue mechanisms, but the rule is not effective in distinguishing among filter mechanisms with different underflow policies, as shown in Section 8.1.

By adding the fuel consumption metrics to make a second selection rule, we are able to distinguish to a greater degree among the performance of filter mechanisms with different underflow policies. Applying this refined selection rule to the traffic simulation results shows that, in most cases, a constant-order extrapolation underflow policy is the preferred filter underflow policy. However, it does identify a few cases where other policies, such as the decaying linear extrapolation model, are better. The results for this selection rule are given in Section 8.2.

We also provide a rule of thumb for selecting gateway mechanisms depending on the inter-arrival characteristics of the enterprise network. This rule is developed by comparing the mean inter-arrival time of various simulation scenarios to the selection rule results (see Section 8.3). The selection guidance rule is: a filter with a constant-order extrapolating underflow policy model should be used by default for most systems. If the mean inter-arrival time of the enterprise network is less than four times the control period of the system, then other underflow policies for filter mechanisms should be tested through simulation. If the mean inter-arrival time is greater than 30 times the control period of the system, then a queue mechanism should be used.

Finally, we show that these selection criteria can be an effective way to improve application performance, yielding improvements of up to 44% in flow metric performance and up to 15% in fuel consumption. The improvement results for all experiments are shown in Section 8.4.

Chapter 9

Conclusion

This research examines gateways between enterprise and embedded networks for applications with real-time, numeric-valued data. It provides insights into how to select gateway mechanisms based on the needs of an application and the characteristics of the network.

9.1 Overview

Chapter 1 provides an introduction to the problem area and a description of the contributions of this research. Chapter 2 provides background on concepts related to the problem and addresses work in related areas that bears on gateway design. Chapter 3 describes how to model gateways as abstract network models and use these models to evaluate gateway mechanisms. Chapter 4 describes gateway mechanisms and policies that can be used to configure them. It also introduces the concept of a filter mechanism that is designed to overcome issues observed in queue mechanisms. Chapter 5 presents a method called Independent Delay Analysis, which provides insights into how network and data characteristics affect the error performance of various mechanisms. Chapter 6 describes the results of evaluating gateway mechanisms using abstract network models. Chapter 7 describes a case study that evaluates gateway mechanisms using simulations of a traffic control application with realistic network models. This chapter also presents the results of the simulations.

Chapter 8 describes selection rules for choosing mechanisms based on the results of the traffic control case study.

9.2 A Workflow for Gateway Design

In this research, we have presented a number of different techniques that address various aspects of the gateway design problem. The results presented are specific to the traffic application we have used as a case study to frame the discussion and provide concrete results. This section organizes the ideas from our research into a workflow that can be followed to obtain similar results for other applications. We also make note of some aspects of the gateway design problem for other scenarios (e.g. embedded-to-enterprise) and types of data (e.g. categorical data). Evaluating mechanisms in these scenarios is outside the scope of the work presented here, but we have made some suggestions that could provide a starting point for future work to expand on what we have presented here.

The general structure of the gateway design workflow is as follows:

1. Identify application characteristics, including characteristics of the data, the network configuration, and performance metrics.

2. Identify candidate mechanisms and policies, using the mechanisms and policies described in our work as a starting point.

3. Explore the behavior of candidate mechanisms using abstract network simulation and analysis methods (e.g. Independent Delay Analysis). This exploration narrows the scope of succeeding, higher cost steps by providing insights into mechanism performance for the application.

4. Evaluate mechanisms with detailed simulations or application testing.

9.2.1 Identify Application Characteristics

There are several aspects of application characteristics to consider. First, we will consider the data streams used by the application and how their characteristics affect gateway design. Second, we consider the properties of the networks used in the application. Finally, we discuss the importance of identifying and selecting application metrics.

9.2.1.1 Data Characteristics

The attributes and characteristics of the application and the data are the first item to address when applying our techniques to other applications. Each data stream in the gateway application must be identified and classified according to the gateway scenario that applies to it: enterprise-to-embedded, embedded-to-enterprise, or embedded-to-embedded (see Section 1.2.1). The scope of our work is limited to the enterprise-to-embedded scenario, but we discuss the other scenarios here briefly to sketch how their design challenges might be addressed.

The design process for other scenarios is similar in the need to identify the data types and develop models for how they change over time, but different because of the kinds of problems that need to be solved. For example, an embedded-to-enterprise data stream arises if we extend the traffic control case study so that the current speed from each vehicle is sent from the real-time network, through the gateway, and over the wireless network to a guidance computer. In the embedded-to-enterprise scenario, mitigating delay caused by the enterprise network is likely to be less important because any delay in the enterprise network will occur after the gateway has sent a message. At that point, the gateway can no longer modify the data or even measure the delay.

The challenge of embedded-to-enterprise scenarios is likely to be dealing with bandwidth and processing constraints. The enterprise network is likely to be bandwidth constrained,

and the guidance computer is likely to be resource constrained. Given these constraints, it would be undesirable to transmit every speed message from the embedded network (potentially hundreds of message per second) on the enterprise network. Thus, the mechanism used in the the embedded-to-enterprise scenario will likely be concerned with how to determine the appropriate speed to send for the lower-bandwidth enterprise application.

For the embedded-to-embedded scenario, a likely case is connecting a high-speed, real-time network to a low-speed, real-time network and transmitting data from the high-speed side to the low-speed side. Similar to the embedded-to-enterprise scenario described above, the challenge becomes one of providing good estimates while adhering to bandwidth constraints.

In addition to the gateway scenario, the type of data in each stream must also be classified according to its characteristics:

- Real-time vs. non-real-time: the extent to which the timeliness of data affects its usefulness. Our analysis deals with real-time speed data, the delivery of which affects the performance of the traffic control application. Any non-real-time data can be delivered eventually (given sufficient available bandwidth). An example of non-real-time data is diagnostic data.

- Time-triggered vs. event-triggered: the semantics of how repeated messages are interpreted by the system. Our approach deals exclusively with time-triggered, periodic data which operates under the assumption that the system is resilient to small perturbations in the data stream. Thus, the estimates used by filter mechanisms to mitigate queue underflow can be assumed to have a small effect on the system. The effect of estimation error may be much larger for event-triggered designs, which may rely on the delivery of every message or of certain messages. It is important to understand

the semantics of the application events — e.g. “at least once” vs “exactly once”. Estimation techniques for event-triggered data (e.g. for filter underflow policy models) are likely to differ significantly from those described in our work. When data has event semantics, repeated transmissions of the event message (where only one actual event occurred) may result in an action that releases energy or allocates resources being repeated to the detriment of the application’s performance.

- Numeric value vs. categorical value: what kind of data the system uses. In this work, we discuss several approaches to estimating numeric data, but most apply only to numeric data. Some data streams may contain categorical data that represent modes, system states, or diagnostic messages. Although categorical data may be represented with numeric values (e.g. enumerated types), categorical values do not have an inherent ordering, so numeric estimation techniques cannot be applied directly and different estimation techniques will be needed. For example, it might be possible to take advantage of application information, such as equivalence classes or which states are reachable from a given state, to estimate categorical values. Alternatively, an ordering might be imposed on the categorical values for the purposes of estimation, if a meaningful ordering can be established and validated.

For the data streams which conform to the enterprise-to-embedded scenario with real-time, numeric-valued, time-triggered data, the approaches described in the rest of this workflow can be applied directly. The implications of handling other types of data should be carefully considered, because the challenges are different for each type, but the rest of the workflow discussion will likely provide a useful starting point for that analysis as well.

9.2.1.2 Network Characteristics

In addition to the characteristics of the application data, network characteristics are also

important to understand because the delays introduced by the network affect mechanism and application performance. The network characteristic that we focus on in the case study is the inter-arrival time of guidance messages at the gateway. There are other ways to characterize the network performance that may be useful for other applications, such as round-trip delay, measures of jitter and offset, and development of probabilistic models for delays.

In order to understand the relationship between characteristics of the networks in the system and the performance of gateway metrics, we apply the selection rule for application metrics (see Chapter 8) to each simulation experiment, then compare those results to the inter-arrival times observed in each simulation experiment. Figures 9.1, 9.2, and 9.3 show probability distribution functions (PDF) of inter-arrival times from a selection of individual simulation runs from the traffic case study. The PDF of the inter-arrival times typically shows a peak around the control frequency for that experiment. The control frequency is 1Hz for all experiments except those in the Slow Update Frequency scenario. For simulations with relatively short delay, such as the Wired Traffic Congestion scenario experiment shown in Figure 9.1, the distribution is close to symmetric around the peak. The guidance packets are emitted from the gateway with a fixed period (e.g. one second). The inter-arrival times that are shorter than the control period are the result of the *previous* packet being delayed, making the current packet's inter-arrival time apparently shorter when it arrives on time.

Two experiments with longer delays from the Wireless Network Congestion scenario are shown in Figures 9.2 and 9.3. Although the peak typically remains at the control frequency, the distribution tends to have more probability density to the right, resulting in higher median values. The inter-arrival times for all experiments with a particular scenario have been collected into a distribution (e.g. all the experiments from a single row in one of Tables

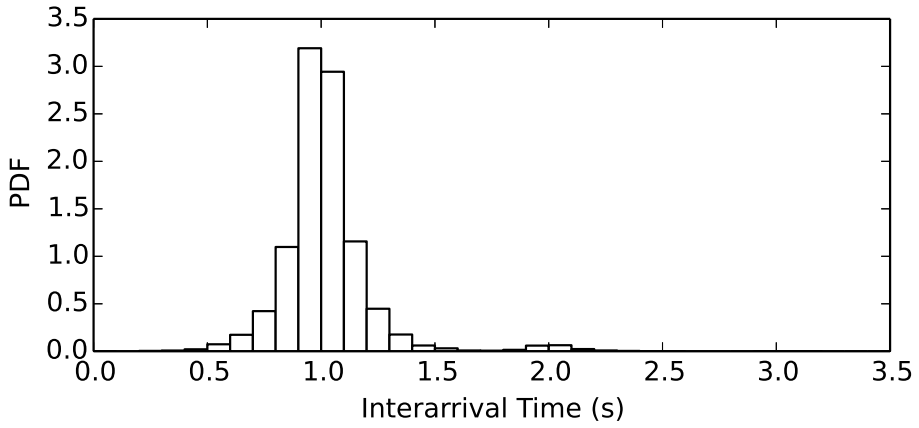


Figure 9.1. Probability Distribution Function of Guidance Packet Interarrivals where Mean Internet Bit Rate=13800; Car Count=30; Random Seed=1033;

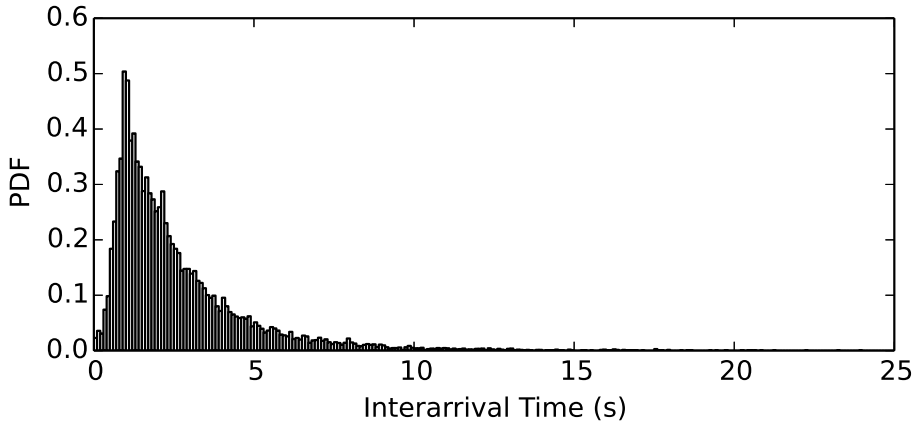


Figure 9.2. Probability Distribution Function of Guidance Packet Interarrivals where Bgnd. Packet Size=1024;Bgnd. Mean Send Period=0.10; Car Count=15; Random Seed=1004;

7.2, 7.3, 7.4, or 7.5). These distributions are presented in Appendix C. They have been summarized using the same boxplots described in Figure 6.1.

From the results discussed in Section 8.3 and presented in Table 8.4, we see that, for this application, the filter mechanisms that estimate delayed values with extrapolation models

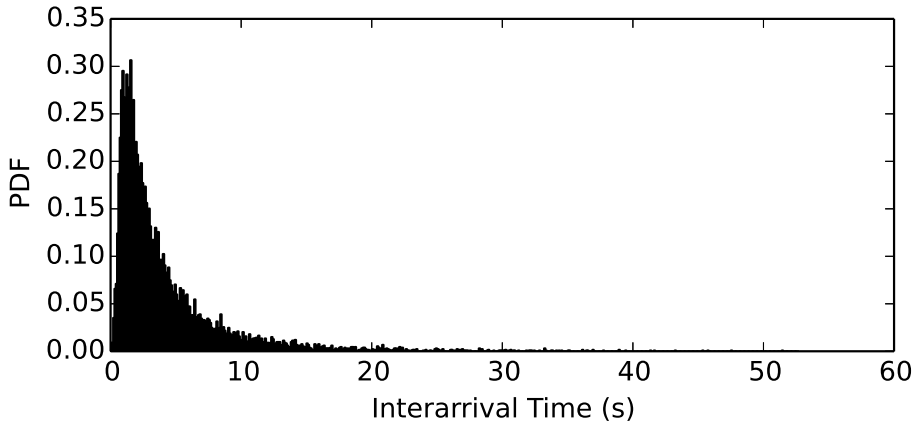


Figure 9.3. Probability Distribution Function of Guidance Packet Inter-arrivals where Bgnd. Packet Size=4096;Bgnd. Mean Send Period=0.10; Car Count=15; Random Seed=1010;

work best when the mean inter-arrival time is close to the control period. Our insight here is that the error for estimation models can become large quickly, especially at times when the guidance velocities are changing rapidly (e.g. due to a perturbation from network delays). Therefore, as delays get longer, the extrapolating models are less effective. We would expect similar results for applications with similar data, although the time constants involved likely depend both on the control frequency of the application (e.g. 1Hz guidance updates) and the time constants of the system (e.g. at a maximum acceleration of $2.5m/s^2$, it takes a vehicle 6 seconds to accelerate from $0m/s$ to a v_{max} of $6m/s$).

9.2.1.3 Application Metrics

In order to compare mechanism performance, there must be one or more metrics that can be measured when various mechanisms are used and the results compared. It is likely that most applications will have one or more metrics that can be used in this way. For example, the metrics we use for the traffic case study are flow, motionless ratio, and fuel consumption (see Section 7.2). These metrics are natural choices that measure aspects

of traffic control problems that designers care about — improving flow and reducing fuel consumption. If similarly useful metrics cannot be identified for another application, we suggest that the application is not sufficiently well-understood and that additional study should be undertaken to understand the behavior of the application before the process of gateway mechanism selection continues.

If an application truly lacks any useful metrics, then the designer could fall back on the MSE between the application data and the gateway output (as described in Section 3.4). This is not desirable because there is no theoretical reason why the MSE should be related to any real-world application performance beyond the idea that the gateway should have as little impact on the system as possible. In particular, evaluations using only MSE might fail to capture regions of instability where even a small error in the gateway output can have a large negative effect on the application. Because of its generality, the MSE metric is useful for exploration with the abstract network simulation, but care should be taken extending those results into real applications without first validating the MSE as a good metric for a particular application.

9.2.2 Identify Candidate Mechanisms and Policies

Once the application characteristics have been explored, then candidate mechanisms and policies should be identified. For the enterprise-to-embedded scenario, the queue and filter mechanisms described in Chapter 4 provide a good starting point. Because they are generic, queues can be used with any application. Although our evaluation has shown the value of application-aware mechanisms, there is no theoretical reason why queues might not perform better for some applications, so they should be included in at least the initial evaluations. Filter mechanisms, in particular those that use the extrapolation models described in Section 4.2.1, should be included also.

The filter mechanisms with extrapolating data models have the advantage of using estimation models that are based only on the data, not on some underlying model of the application, so they can be used with any numeric-valued application data. Other filter mechanisms should be evaluated, but their selection is based on the designer's insight into the characteristics of the data and the application. Another way application knowledge can be applied is to take advantage of state in the application that can be inferred from the data. For example, a gateway might be able to distinguish traffic conditions (congested vs. free flowing) from guidance data and adjust its estimation accordingly.

For other scenarios and data types, similar approaches for identifying mechanisms are possible. For state estimation problems, it is also likely that there are existing algorithms and techniques (e.g. Kalman filters) that can be applied as gateway mechanisms.

Another way to identify mechanisms is to find issues with existing mechanisms and develop new mechanisms to counteract them, as we have done with filter mechanisms, which mitigate underflow problems observed in queues. Exploration of the behavior of mechanisms can be useful to provide these insights. The ideas of exploring mechanism behavior are described in more detail in the next section.

9.2.3 Explore Candidate Mechanism and Policy Behavior

Abstract network simulation and Independent Delay Analysis are two approaches we have used in this work to explore the mechanism and policy design space. This section describes some of the tasks and ideas related to exploration of the mechanism properties and performance. The purpose of this exploration is to understand how application data and timing interacts with different mechanisms and policies. These techniques can be used to identify new mechanisms or policies that may perform better than existing ones. They may also be

used to narrow the scope of more expensive evaluation methods to only the most promising mechanisms and policies.

9.2.3.1 Application Data

In order to apply these exploratory techniques, one of the first tasks is to obtain one or more data sets that are representative of data from the application. Ideally, these data sets should be collected directly from the application or a detailed simulation such as the OPNET® simulation used in the traffic case study. It is important that these data sets cover a wide range of operating conditions of the application, lest the designer optimize the gateway for a narrow case, only to find out that the application only behaves that way a small percentage of the time. For categorical data, data sets should encompass all possible values (if feasible). Some values may be unreachable under fault-free conditions (e.g. error states), so fault injection might be needed in the application [54]. For event-triggered applications data, it is important to ensure that the data set includes rare events, otherwise they may never be considered in the estimation approaches. For extremely rare events, some artificial manipulation of the data may be required. Care should be taken that the dataset remains representative.

Although the ideal case is to obtain samples directly from the application data, this may not always be possible. The application may not have been implemented yet, or the scope of the effort required to operate the system for the purposes of obtaining data may be too large, in which case some substitute source of data is needed.

We first faced the challenge regarding the availability of representative data when we began our exploration of gateway design with abstract network models. We chose to focus on systems with real-time, time-triggered data since gateway systems with these characteristics are already being used. We collected vehicle speed data using the OBD-II diagnostic port (see Section 3.3) to provide representative data for a time-triggered application. The

vehicle speed data was used in the abstract network simulations which led us to develop the filter mechanism to mitigate queue underflow. We then began developing the traffic case study to validate our abstract network simulation results. Even though the vehicle speed data is not perfectly representative of data we observe in the traffic case study, we were still able to obtain useful results from the abstract network simulations.

9.2.3.2 Abstract Network Simulation

The speed data we have obtained from the OBD-II apparatus is quite different from the guidance data observed in the traffic case study. Another limitation of the speed data is that there is no control application, so there is no timing information. If we obtain data directly from the traffic simulation (or any other "real" implementation of an application), it can also include the arrival timing that was observed.

Fortunately, for the abstract network simulation technique, timing data is provided by the network models, and the input data values are all that is needed (although they may need to be resampled or interpolated to the periods used in the abstract network simulation). For our application, the abstract network simulation provides timing data for the enterprise network model in the form of a Poisson process. The Poisson process is not a realistic representation of any IP or wireless network. In particular, we later found that the characteristics of the inter-arrivals for periodically transmitted data follow a distribution like that shown in Figures 9.1, 9.2, and 9.3. This two-sided distribution reflects the fact that the periodically transmitted control data is perturbed by the network, but the peak remains at the control period. The distribution is two-sided because the values measured are inter-arrival times, so a "late" packet makes the following packet look "early" when it arrives on time.

Another way that the abstract simulation is limited is the use of the MSE metric to compare mechanisms (see Section 3.4). As we have already discussed in Section 9.2.1.3,

the MSE does not have any general relationship with application performance — whether MSE is representative of application performance depends on the application.

Despite these limitations, the abstract network simulation technique has proven useful in obtaining insights into the way that gateway mechanisms behave in the enterprise-to-embedded scenario. The simple network models have fewer parameters (compared to the traffic case study simulations). Fewer parameters means fewer experiments to cover the parameter space, and (in this case), simulations that are significantly faster to run than the OPNET® simulations for the case study. The primary insight we gained through the abstract network simulations was that underflow in queue mechanisms was the source of much of the input-output error we observed. This insight led us to the development of filter mechanisms, which demonstrate improved error performance in the abstract network simulations.

It is important to note that the traffic case study bears out the results from the abstract network simulation *in this case*, but there is no fundamental reason why that should be so. It is possible that we would have obtained different results from the case study, finding shortcomings in our early results because the models were not representative. If this had been the case, the abstract network simulations could have been made more useful by developing a more representative timing model for the enterprise network. We have found the abstract network simulation technique to be useful in our work and believe it would be a useful general-purpose tool as long as its limitations are considered in the evaluation of results.

If the application data has characteristics other than time-triggered, numeric-valued data, some changes may be needed. For categorical data, a metric other than MSE is needed because the "error" between two categorical values is not well-defined. For some applications, it may be possible to develop a definition for error. For example, a pair-wise assignment

of error weights could define the "badness" of estimating the first state when the second state is the true state. Defining error metrics for event data may require similar arbitrary assignments. When developing notions of error (or another general metric) for categorical and event data, care should be taken to avoid introducing bias into the results with arbitrary assignments.

The abstract network simulation can be adapted to the study of other gateway scenarios by changing the network models, but some care must be taken. For example, in the embedded-to-enterprise scenario, the enterprise network is now the receiving network, so there is no longer a real-time driver on the queue output. This may require re-defining mechanisms to include a built-in periodic service interval or the implementation of a poll request from the data destination in the enterprise network, depending on the needs of the application.

9.2.3.3 Mechanism-Specific Metrics

In Section 9.2.1.3 above, we discussed application metrics and how they can be used to compare the performance of mechanisms in an application. But for exploring the behavior of mechanisms, there may also be metrics related to the structure of the mechanisms themselves. For example, in our evaluation of queue mechanisms with abstract network simulations (see Section 6.1), there are several metrics that describe the structure and behavior of the queue, e.g. queue length and number of dropped messages. These metrics provide insight into how the mechanism behaves under various workloads and can lead to a deeper understanding of the mechanism, even if they do not give insight into how the mechanism performs relative to other types of mechanisms. This may lead the designer to ideas for improvements similar to our development of filter mechanisms based on the underflow insights gained from studying the behavior of queue mechanisms.

9.2.3.4 Independent Delay Analysis

Another useful tool for exploring mechanism and policy performance in enterprise-to-embedded scenarios with numeric-valued data is the Independent Delay Analysis (see Chapter 5). This method is useful because it provides a network-independent way to understand the effects of delay on different filter underflow policy models. It provides insights that can be used to develop new policies similar to our development of the Decaying Linear Extrapolation model (see Section 5.3).

For applications that conform to other gateway scenarios, these approaches may not be as useful because delay may no longer be a primary driver behind the error introduced by the gateway. For example, in the embedded-to-enterprise scenario, the delay in the enterprise network occurs after the gateway has already sent the message — it may not even be observable by the gateway.

9.2.4 Evaluate Mechanisms with Detailed Simulations

Depending on the goals of the gateway designer, the exploration techniques in Section 9.2.3 may or may not be useful. Ultimately, the approach we have described for exploring mechanisms and policies is an empirical one — it must be validated by testing mechanism performance with the application. Certainly, exploration of mechanisms and policies presents many opportunities to extend our work into other scenarios and data types and to further understand how gateways may be used for internetworked systems. However, if the goal of the designer is simply to evaluate known mechanisms and policies for a particular application, then exploration is not needed if an application or a detailed simulation is available to test the performance of different mechanisms.

We used the traffic simulation as a case study to simulate a variety of wireless and wired network scenarios for the guidance applications described in [8]. The OPNET® simulation

and the well-known traffic flow model combine to provide a realistic simulation environment that we use as a proxy for a "real" application. Evaluating mechanisms with a detailed simulation provides more authoritative guidance about the mechanism selection for the particular application (see Section 8.3 for our results).

Even if the application being studied is already available, we suggest that mechanisms first be evaluated with a highly detailed simulation system (such as the OPNET® simulator). The simulation environment provides repeatability and observability that may be difficult to obtain in the implemented application due to variations in equipment function and the test environment. Given the costs of running a real implementation, a simulation approach also allows a broader scope of inquiry since the cost of individual simulation runs is likely to be lower than experimental trials with the application. When possible, simulation results should be validated with the actual application to demonstrate that the results are not just an artifact of the simulation. In our work, no real implementation was available to us. Implementing the traffic control application described in Chapter 7 would have required obtaining and outfitting vehicles for control, developing wireless infrastructure, and obtaining and paying for fuel, drivers and track time to run the experiments. These costs would have far exceeded the resources available for this project.

In our evaluation of mechanisms with the OPNET® simulation, we conducted 22,080 simulation runs (see Tables 7.2, 7.3, 7.4, and 7.5). These simulations took from a few minutes to a few hours each to run and generated hundreds of gigabytes of data. All that effort was needed to produce the results for just eight mechanisms across 18 different environment scenarios. If only the two mechanisms studied (queues and filters) are considered, the policies for each mechanism provide hundreds of possible configurations. It would not have been feasible to test them all using the OPNET® simulation. For this reason, we sug-

gest that the abstract network simulation approach be retained because it can reduce the number of mechanisms that need to be tested in the detailed simulation.

9.2.5 Workflow Summary

In this section, we have described how the different aspects of this research fit into a workflow that has been effective in evaluating gateway mechanisms and policies in our research. We believe it could be applied to other, similar applications to identify and evaluate appropriate gateway mechanisms. Although the scope of our work is limited to enterprise-to-embedded gateways with time-triggered, numeric-valued data, we have also suggested some of the ways this workflow might be adapted to other scenarios and data types. In the following section, we will restate the results of our research in terms of the contributions we first outlined in Section 1.4.

9.3 Research Contributions

As technologies emerge that connect embedded systems with enterprise systems, solutions are needed that can be used to manage data flows between heterogeneous networks. Gateway architectures are one such solution. We have identified mechanisms and policies that can be used to manage these flows for the particular case of enterprise-to-embedded networks using numeric-valued, time-triggered data flows. Through abstract simulation evaluation and a case study, we have compared the performance of these mechanisms and policies. Furthermore, we have identified a selection rule for choosing mechanisms based on the measurement of application metrics in the traffic simulation. Using this rule and the inter-arrival characteristics of each simulation scenario, we have provided selection guidance for the case study application. Although these contributions are offered in the context

of the applications described here — the driving data used for abstract network simulations and the traffic case study simulations run in OPNET® Modeler — the workflow we use to analyze various mechanisms can be applied to other, similar applications to obtain similar results.

9.3.1 Identify Gateway Mechanisms and Policies

This research identifies mechanisms and policies that can be used to mitigate problems that arise in a gateway.

In this work, we collect mechanisms and policies from several existing literature and use them to develop a set of mechanisms and policies that can be used in gateway applications. Both queue mechanisms and filter mechanisms are examined, and for each mechanism, we identify a number of policies that can be used to configure them and show which policies are useful for tuning mechanisms to improve gateway performance.

Queue mechanisms are already being used to manage data flows in Internet routers, so we build on this work by identifying policies that have their roots in prior work on queue management. The three policies for configuring queues are: queue length, queue overflow, and queue underflow. Queue length specifies how long the queue is allowed to grow, while queue overflow defines which message is removed from the queue to keep it from exceeding this length. Queue underflow specifies the behavior of the queue when a message value is desired, but the queue is empty.

Queue underflow turns out to be particularly interesting in the enterprise-to-embedded scenario. Because of timing mismatches between the enterprise and embedded networks, the queue can underflow. When this happens, succeeding messages are delayed because of missed time slots on the embedded network. Based on insights gained from abstract network simulations, we have designed the filter mechanism to mitigate these underflow

problems. When a message is delayed, a filter mechanism sends an estimated value on the embedded network and then drops the incoming, delayed message when it arrives. This improvement (over queues) eliminates the underflow delay.

Filter mechanisms have just one policy: filter underflow policy. However, unlike queues, which are generic, filters are application-aware, so setting this policy involves choosing a data model to make estimates for delayed messages. The most basic of these policies is the constant-order extrapolation filter, which resends the last observed value. More complex models evaluated in this research involve extrapolation and hybrid extrapolation algorithms. However, many other data models are possible, depending on the needs of the application.

Filter and queue mechanisms and their policies are described more fully in Chapter 4.

9.3.2 Demonstrate Performance Advantages of Application-Aware Mechanisms

This research demonstrates that application-aware mechanisms exhibit improved performance when compared with generic mechanisms in the enterprise-to-embedded gateway applications studied.

Queue mechanisms are generic mechanisms in the sense that they do not use information about the content of messages to determine how messages are processed. The FIFO queues studied here merely emit messages in the same order they are sent (except for dropped messages). Filter mechanisms are application-aware: they must be aware that they are processing periodic data, and they must make estimates for delayed messages during underflow.

Both the abstract network simulation results and the traffic case study show that the application-aware mechanisms provide improved performance over queue mechanisms. In abstract network simulations, we observe that the character of the mean squared error dis-

tribution for filter mechanisms is distinctly different from queue mechanisms and biased toward lower error when compared with queue mechanisms using the vehicle speed application data we collected (see Chapter 6). In the traffic case study, filter mechanisms exhibit the same or better performance in all but a few cases (see Chapter 7). Using the selection criteria from Chapter 8, we measured performance improvements of up to 44% in flow metric performance and up to 15% in fuel consumption. These improvements are specific to the traffic application, but we expect similar results would be obtained for applications with similar data characteristics.

9.3.3 Provide Selection Guidance for Gateway Mechanisms and Policies

This research uses simulation evaluation results to give guidance for selecting the appropriate mechanism and policy for a particular scenario.

There are three aspects to the mechanism and policy guidance developed in this research: (a) comparing policies for particular mechanisms, (b) using analysis methods to select among mechanisms and develop new mechanisms, and (c) defining a selection rule for the traffic case study based on the OPNET® simulation results.

First, we offer insight into the effectiveness of policies in queue and filter mechanisms. We show that varying queue policies has little effect on performance, but filter policies can improve performance in cases where mean inter-arrival times on the network are less than four times the control frequency.

Through abstract network simulations with the vehicle speed data as input (see Chapter 3), we determine that no queue length policy has a consistent effect on mechanism performance for all the data sets we examined. Furthermore, we show that the overflow policy has no useful effect on the performance of queue mechanisms. Queue underflow policies

are not examined because they are effectively equivalent for the enterprise-to-embedded scenario. All the queues studied here use the mailbox underflow policy because that is the most reasonable choice, even though a typical enterprise system that uses queue mechanisms would employ a send-no-value underflow policy. The insight that queue policies have little effect on queue performance is born out in the case study, in which all the queue policy combinations exhibit effectively the same performance. There was no scenario where the selection rules selected any particular queues of the four queues tested — either all were selected, or none were (see Chapter 7).

Similarly, we compare filter underflow policies using the abstract network simulations and the vehicle speed data. The results show that Decaying Linear Extrapolation models improve performance over the basic, constant-order extrapolation model. Higher-order extrapolation models are never an improvement with the data sets tested (see Chapter 6). The traffic case study results also show that there are some cases where Decaying Linear Extrapolation policy models show improved performance (see Chapter 7).

Second, the Independent Delay Analysis method described in Chapter 5 can give additional insights into choosing models for filter underflow policies and designing new models. The abstract network analysis results from Chapter 6 show that the results for filters with various underflow policies are consistent with the results from the Independent Delay Analysis. Furthermore, the traffic case study in Chapter 7 selects the Decaying Linear Extrapolation model as the preferred policy for filter mechanisms in some experiments, and this policy model has been developed using the Independent Delay Analysis.

Finally, we discuss our insights into mechanism selection for the traffic case study. The mechanism selection guidance for the traffic control application we studied is:

- A filter with a constant-order extrapolating underflow policy model should be used by default for most systems.

- If the mean inter-arrival time of the enterprise network is faster than four times the control period of the system, then other underflow policies for filter mechanisms should be tested through simulation.

- If the mean inter-arrival time is slower than 30 times the control period of the system, then a queue mechanism should be used.

Chapter 8 describes how this guidance is developed based on the simulation results from this research. Further research is needed before this guidance can be generalized to other applications. However, we believe our results provide some useful insights that are likely to apply to other applications. First and foremost, filter mechanisms are likely to perform better than queues in similar enterprise-to-embedded scenarios, regardless of the data. Because the constant-order extrapolating filter simply resends the most recent value, the resulting mechanism is similar to a queue with a mailbox underflow policy, *with the important difference that the filter mechanism discards delayed packets*. The fact that the constant-order extrapolating filter mechanism is selected for a majority of the traffic application results suggests that this single innovation — dropping delayed packets — is useful for enterprise-to-embedded gateways, regardless of whether more sophisticated estimation models are used for the filter underflow policy. Second, we observe that the traffic control scenarios that had inter-arrival times close to one second (the period of the guidance packets sent by the application) also tended to select the filters with more sophisticated estimation models because these models are most effective for systems with short delays.

The rule stated above can be given a more general form: *use filters with non-constant extrapolating models for networks faster than physical time constants; use filters with a constant extrapolation model for networks that are about the same speed as the physical time constant of the system; and use queues for networks that are much slower than the*

physical time constants of the system. Based on the evaluations we have conducted with the traffic application, we believe that this rule is likely to hold for other, similar applications.

9.4 Future work

In the process of conducting this research, we have identified several possible areas where this work might be extended.

Additional case studies would help to broaden the understanding of how the values of the data being used in the system affect the performance and selection criteria for application-aware mechanisms. It would be interesting to compare open-loop and closed-loop applications. Since the traffic control algorithm has a feedback loop, it corrects for errors and delays induced by the network and the gateway mechanisms. An open-loop system would likely show the differences between various mechanisms more clearly and provide additional insight, just as the abstract network simulations did. Examining open-loop systems would also provide an opportunity to enrich and extend the abstract network simulation approach.

For closed-loop systems, closing the loop through another gateway mechanism would open up inquiries into the embedded-to-enterprise scenarios. But it would also provide an opportunity to study the interactions between gateway mechanisms that do not have an explicit communication channel, but are linked by the feedback loop.

Expanding the range of application-aware mechanisms also seems promising. In particular, machine-learning approaches could be used to train gateway mechanisms. In the context of an application like the traffic control case study, the metrics and simulation framework provide ready-made feedback for supervised learning techniques. The performance of mechanisms based on learning systems would likely improve over time. Instead of trying to train the mechanisms ahead of time, untrained vehicles could be allowed to enter

and leave the system, although allowing vehicles to enter and leave the simulation would require increasing the scope of the traffic simulation beyond the ring-road scenario. As untrained vehicles' learning models were exposed to data, their performance would likely improve. This is well-suited to a traffic scenario because most drivers repeat similar routes and schedules (e.g. commuters).

Another avenue to explore would be to study the security implications of gateways and mitigations for malicious attacks. For example, an attacker could try to decrease fuel economy by varying guidance data in a way that caused unnecessary acceleration and deceleration. An attacker could also try to destabilize engine control loops or adversely affect battery charge cycles in electric or hybrid vehicles.

Appendix A

Abstract Network Model Evaluations

This appendix provides the extended results for the evaluation of queue and filter mechanisms using abstract network models as described in Chapter 3. These results are a continuation of the results in Chapter 6.

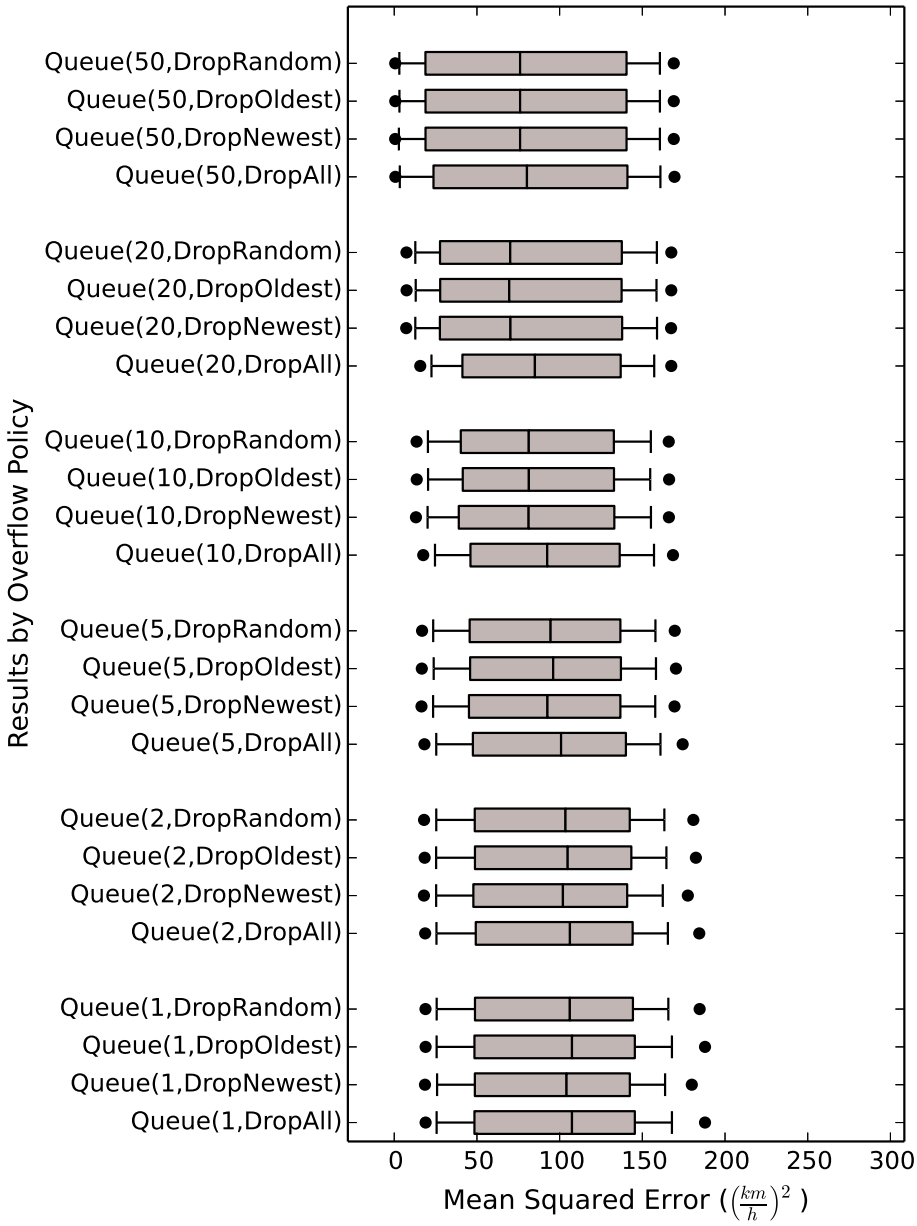


Figure A.1. MSE for Various Overflow Policies grouped by Queue Length (Monroeville I Data Set)

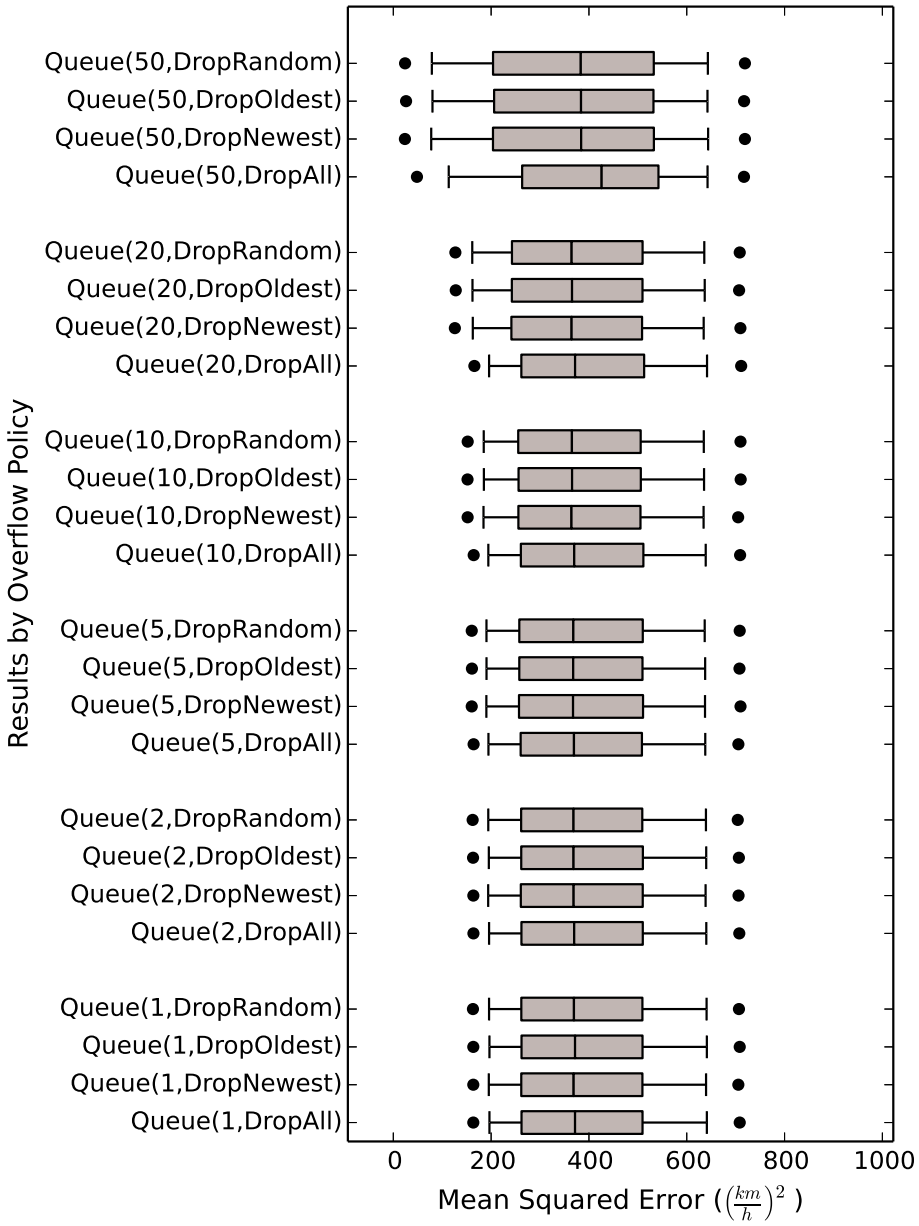


Figure A.2. MSE for Various Overflow Policies grouped by Queue Length (Beechwood Data Set)

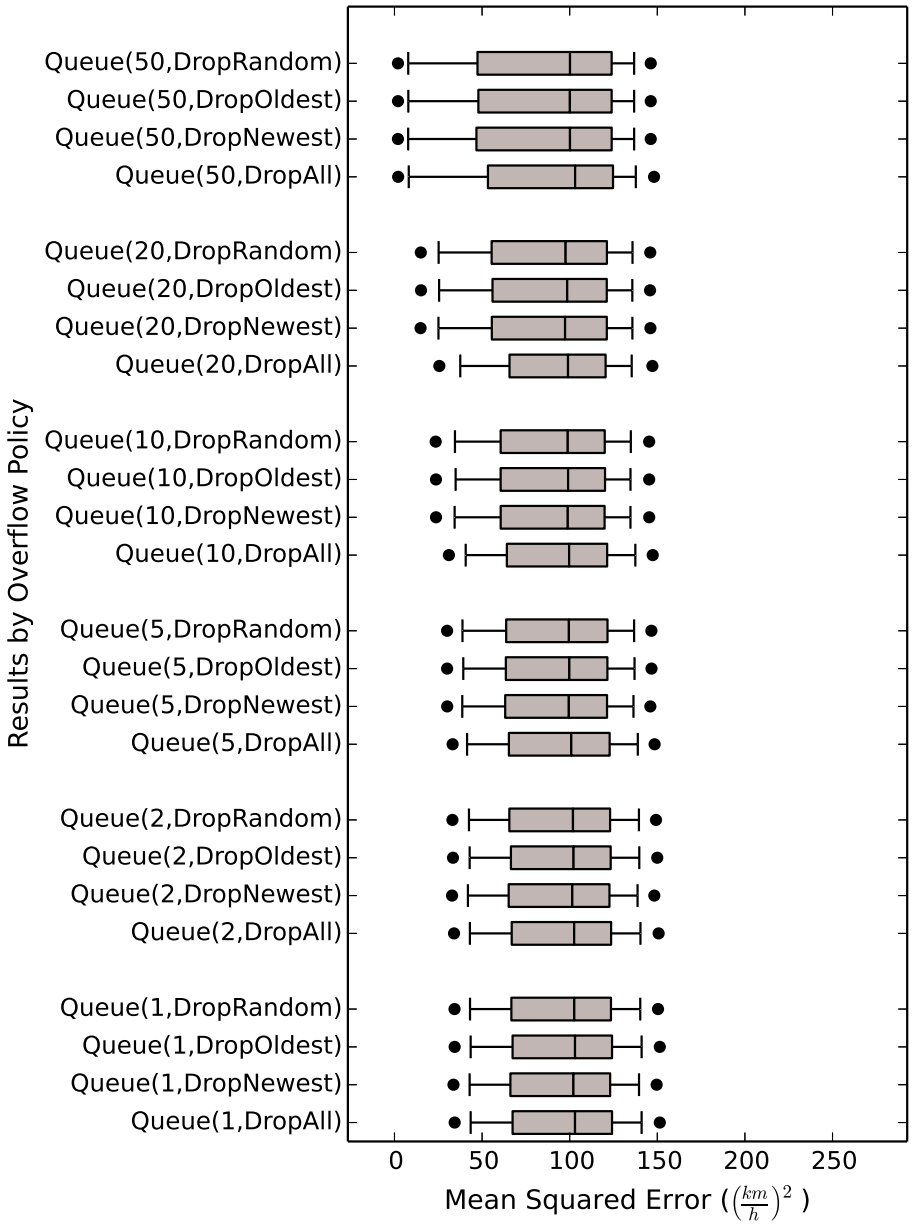


Figure A.3. MSE for Various Overflow Policies grouped by Queue Length (Monroeville II Data Set)

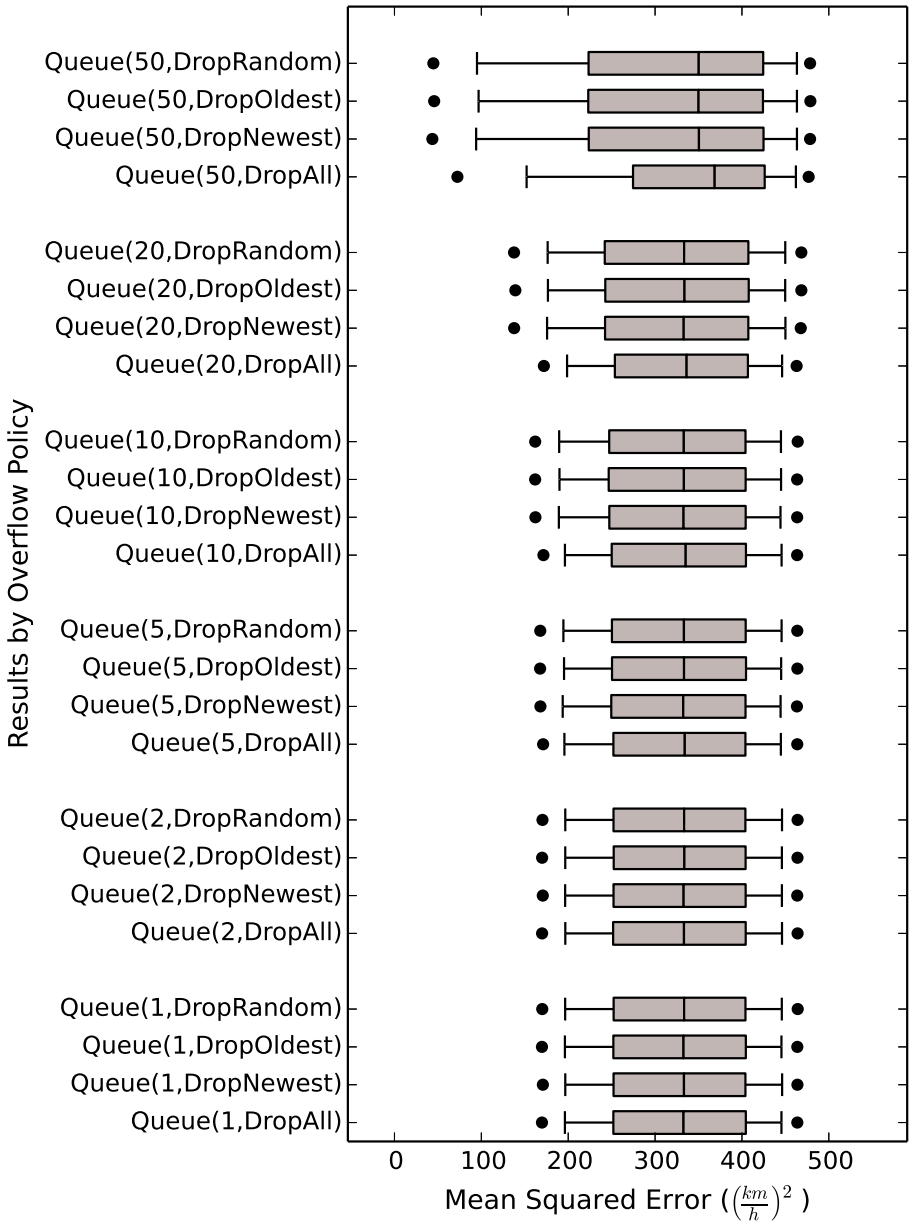


Figure A.4. MSE for Various Overflow Policies grouped by Queue Length (Squirrel Hill Data Set)

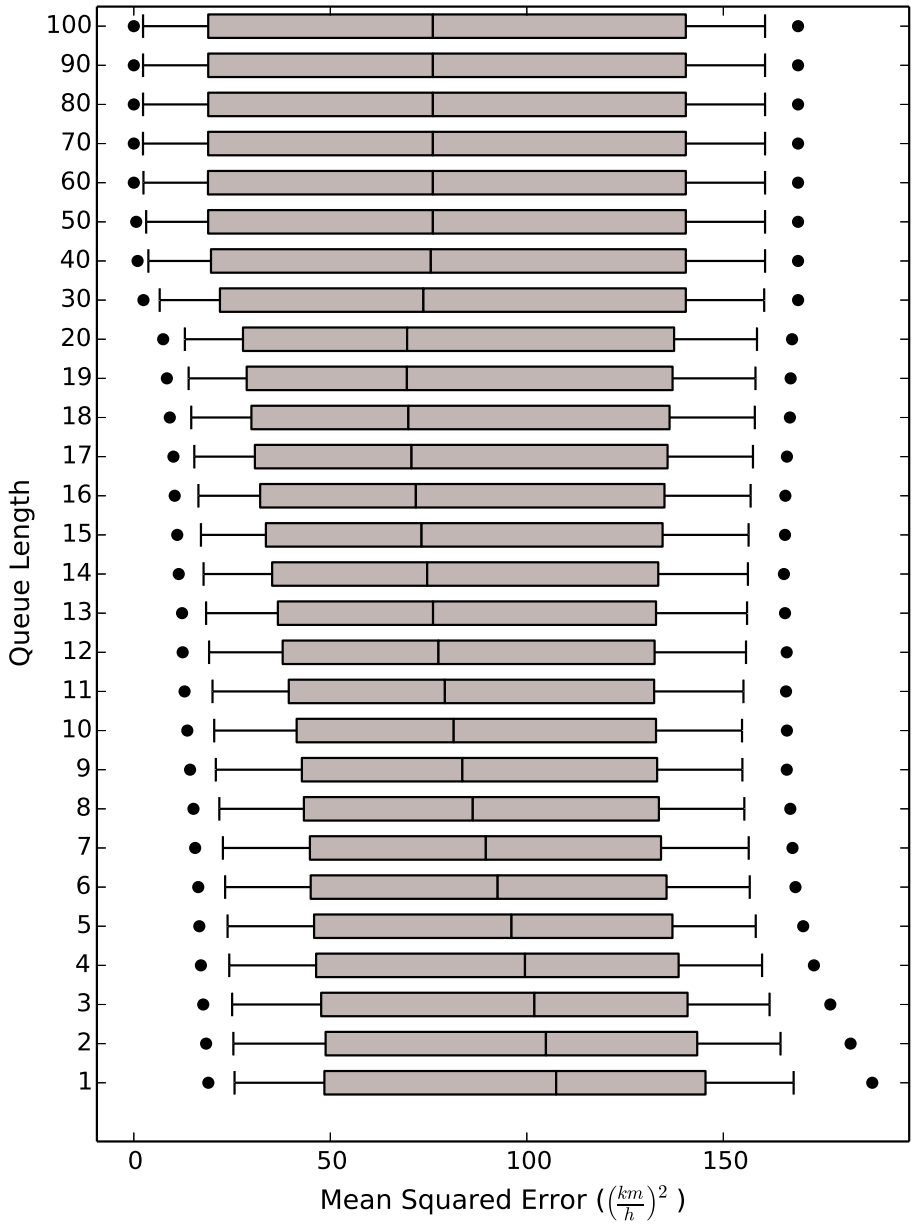


Figure A.5. MSE for Various Queue Lengths (DropOldest, Monroeville I Data Set)

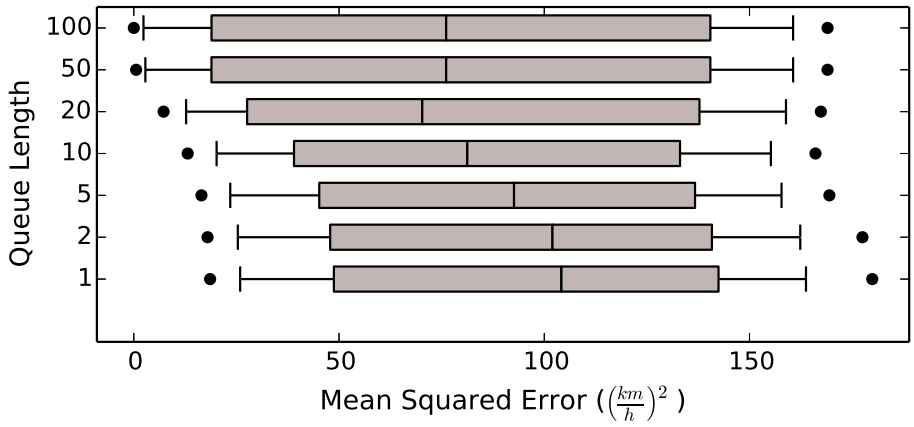


Figure A.6. MSE for Various Queue Lengths (DropNewest, Monroeville I Data Set)

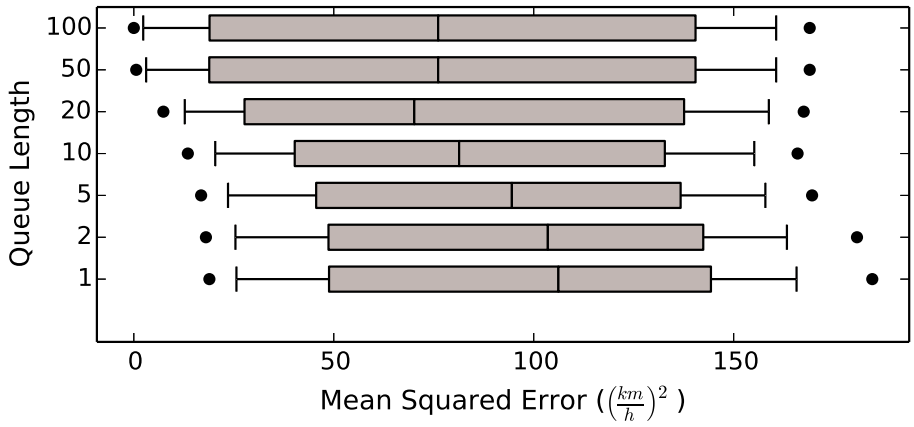


Figure A.7. MSE for Various Queue Lengths (DropRandom, Monroeville I Data Set)

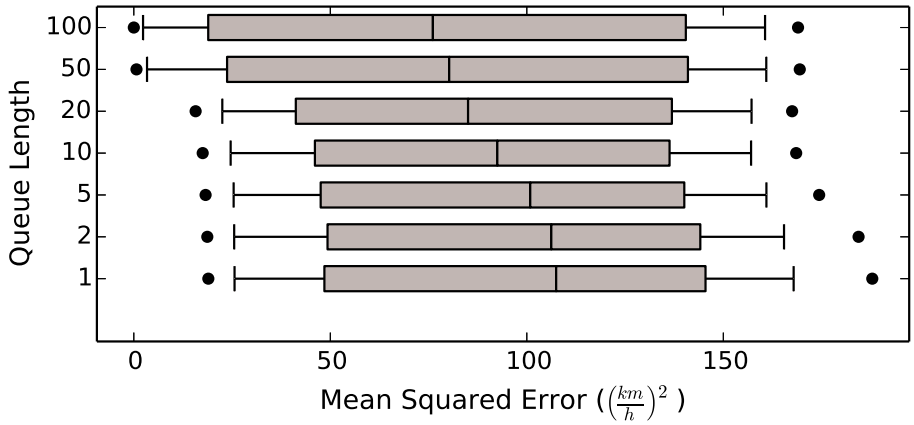


Figure A.8. MSE for Various Queue Lengths (DropAll, Monroeville I Data Set)

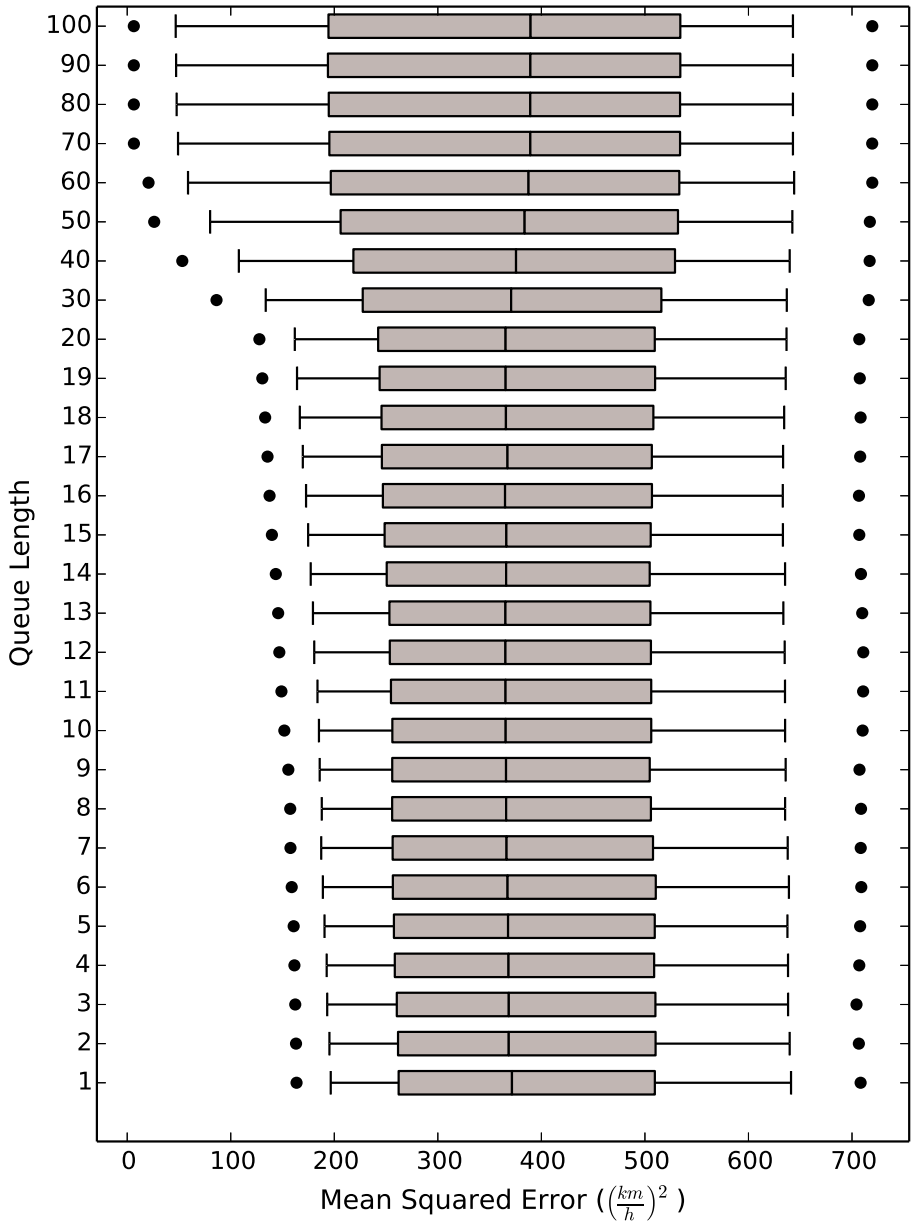


Figure A.9. MSE for Various Queue Lengths (DropOldest, Beechwood Data Set)

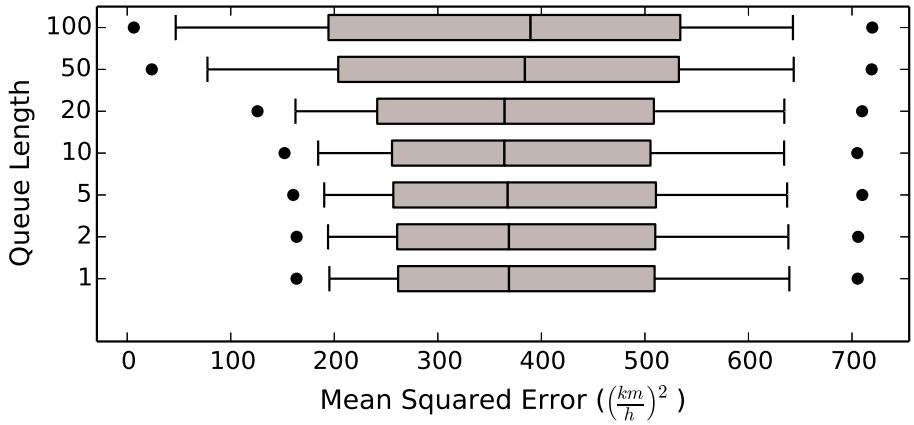


Figure A.10. MSE for Various Queue Lengths (DropNewest, Beechwood Data Set)

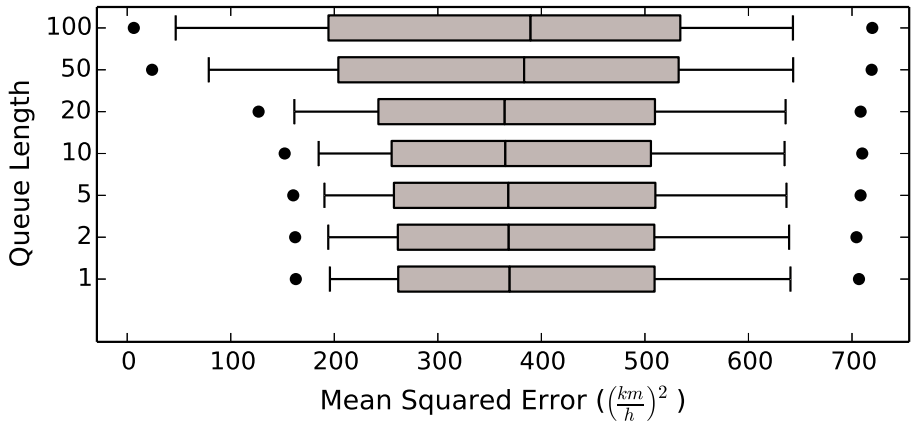


Figure A.11. MSE for Various Queue Lengths (DropRandom, Beechwood Data Set)

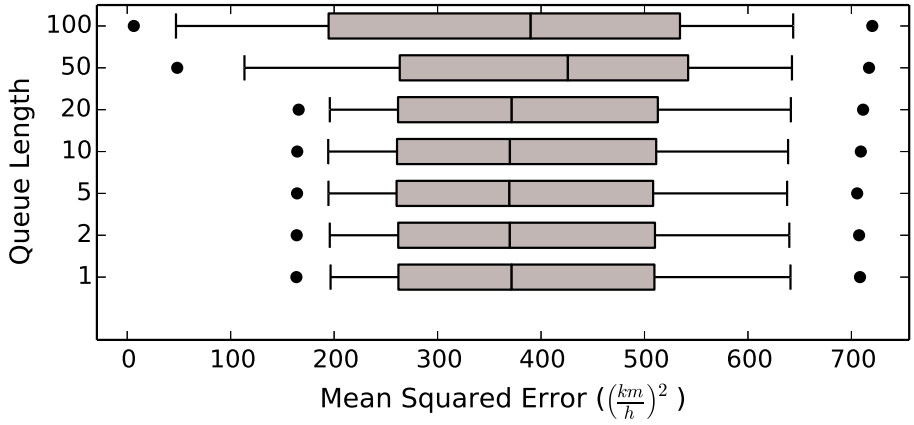


Figure A.12. MSE for Various Queue Lengths (DropAll, Beechwood Data Set)

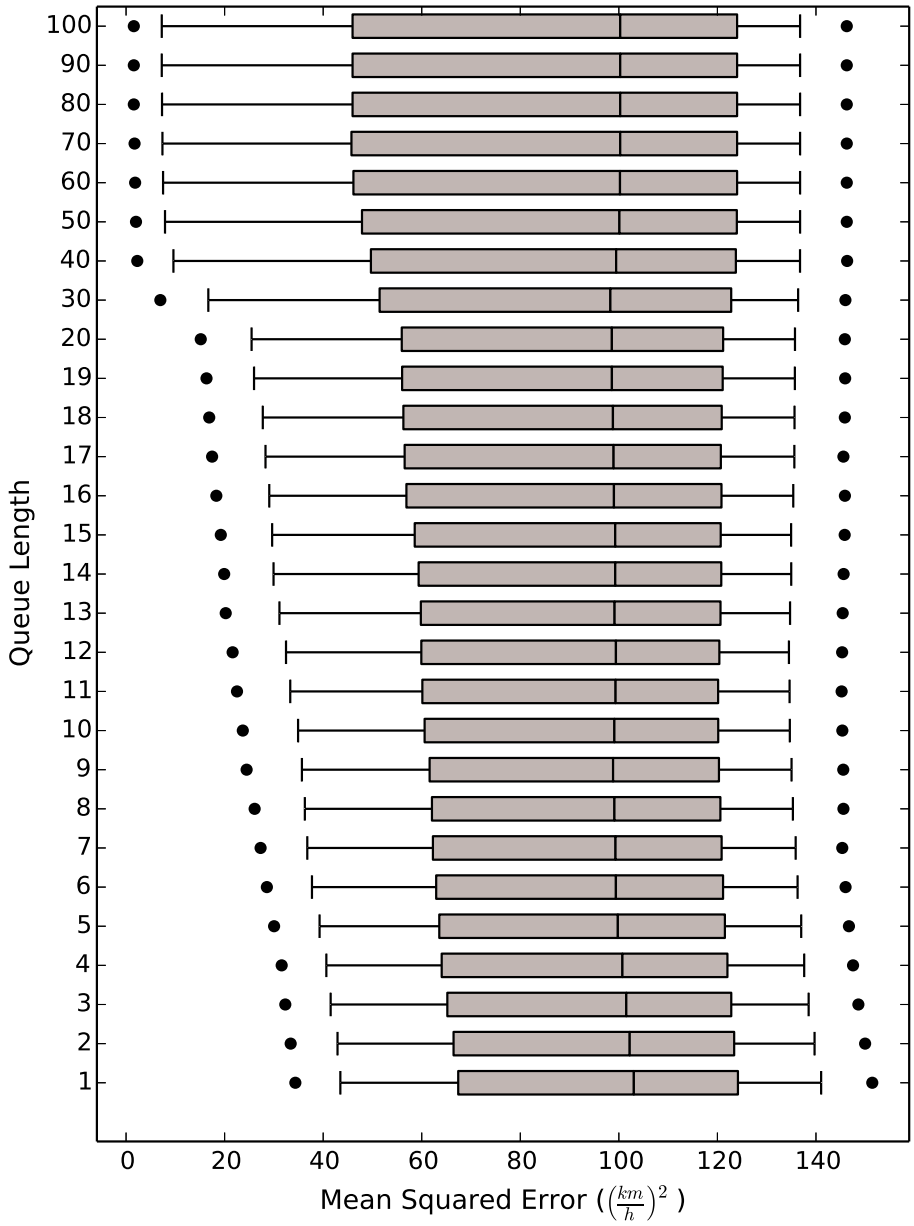


Figure A.13. MSE for Various Queue Lengths (DropOldest, Monroeville II Data Set)

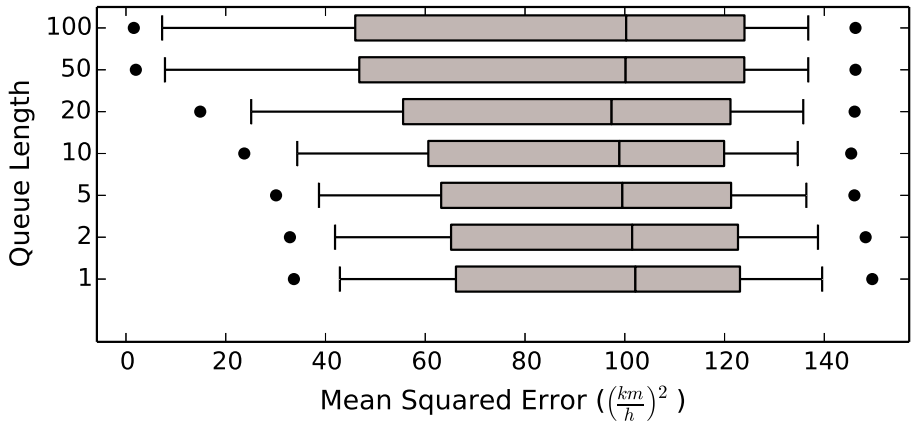


Figure A.14. MSE for Various Queue Lengths (DropNewest, Monroeville II Data Set)

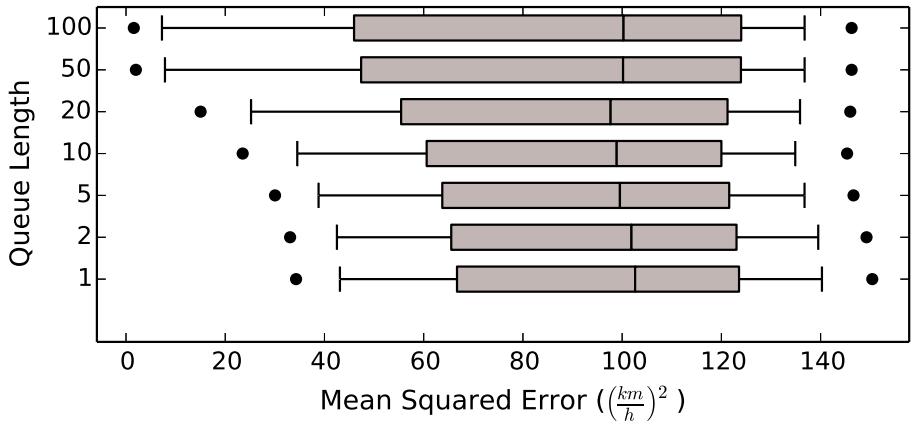


Figure A.15. MSE for Various Queue Lengths (DropRandom, Monroeville II Data Set)

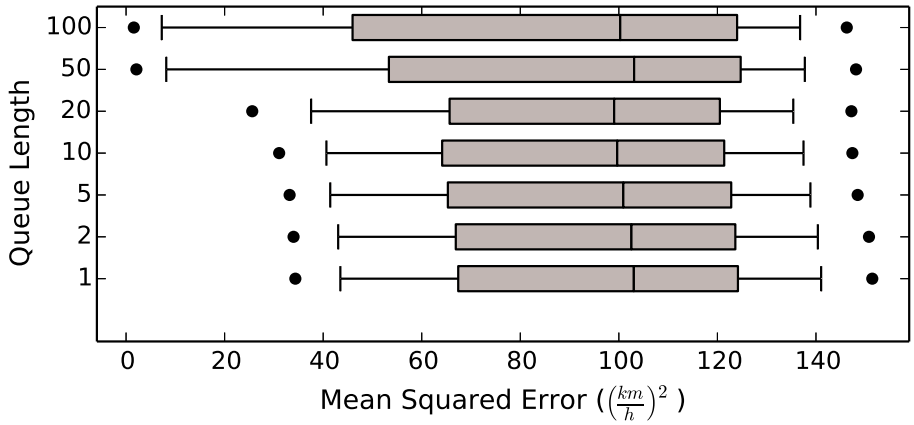


Figure A.16. MSE for Various Queue Lengths (DropAll, Monroeville II Data Set)

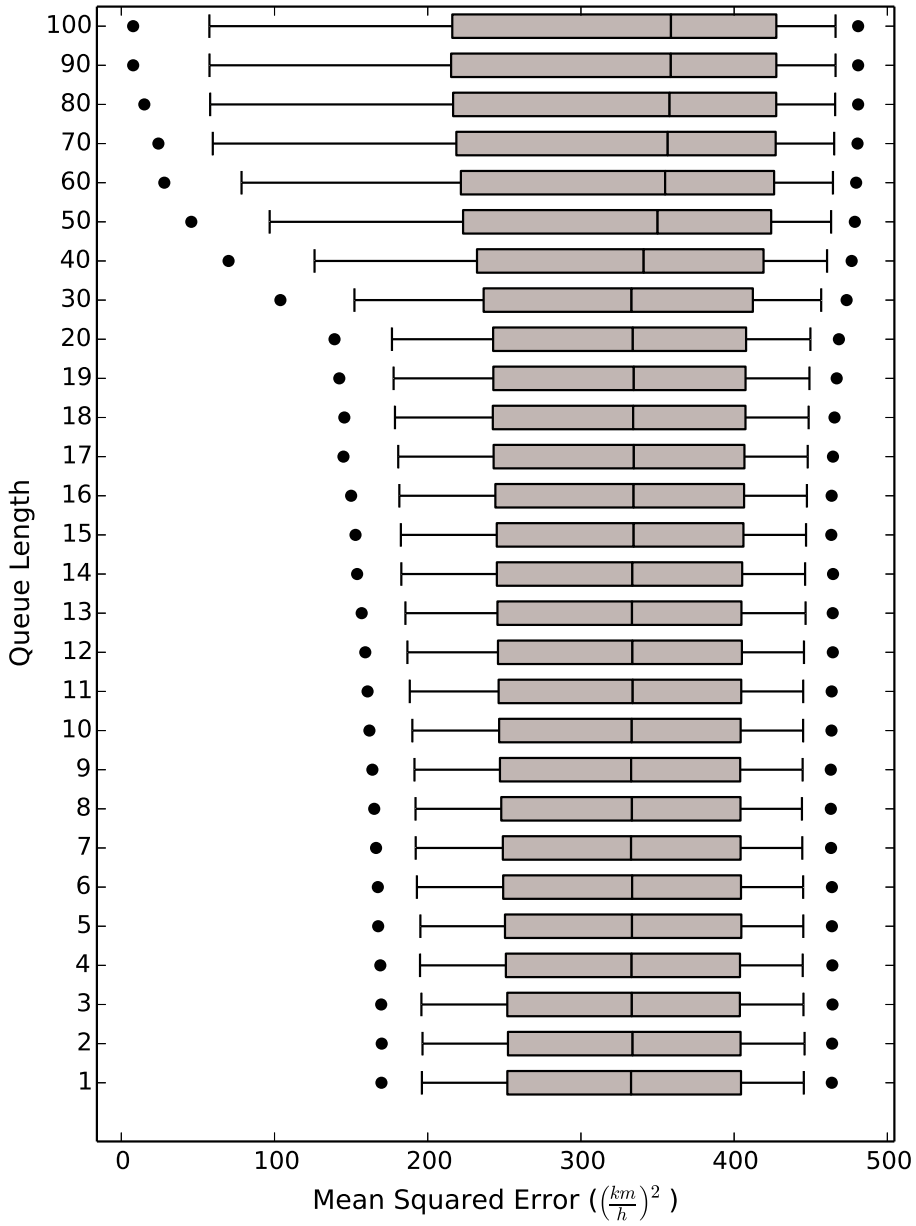


Figure A.17. MSE for Various Queue Lengths (DropOldest, Squirrel Hill Data Set)

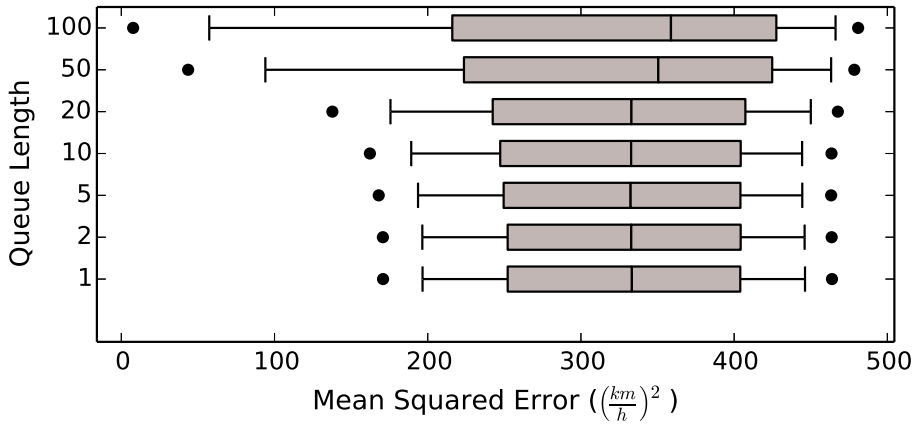


Figure A.18. MSE for Various Queue Lengths (DropNewest, Squirrel Hill Data Set)

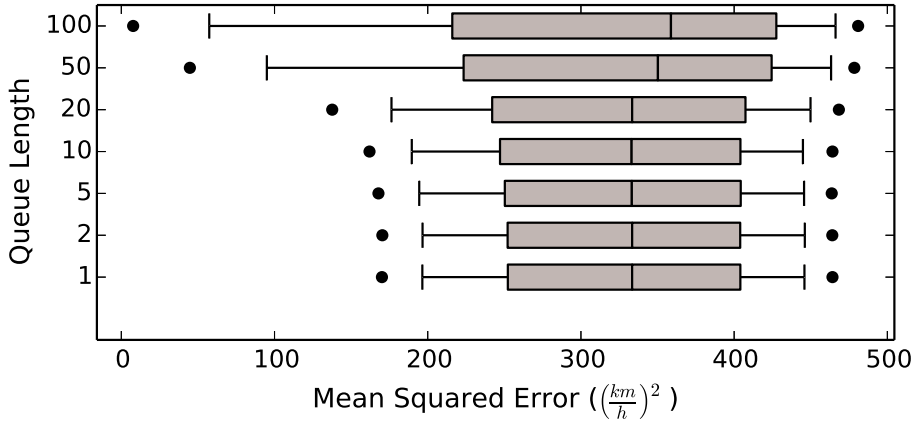


Figure A.19. MSE for Various Queue Lengths (DropRandom, Squirrel Hill Data Set)

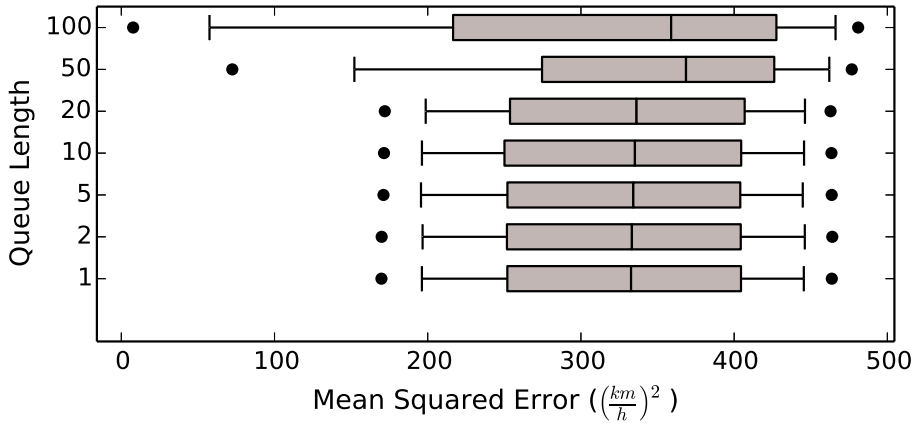


Figure A.20. MSE for Various Queue Lengths (DropAll, Squirrel Hill Data Set)

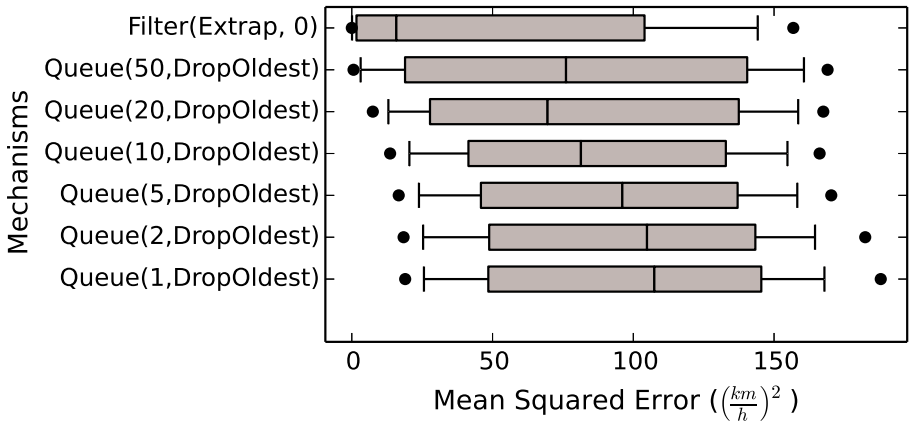


Figure A.21. MSE Comparison for Queue and Filter Mechanisms (Monroeville I Data Set)

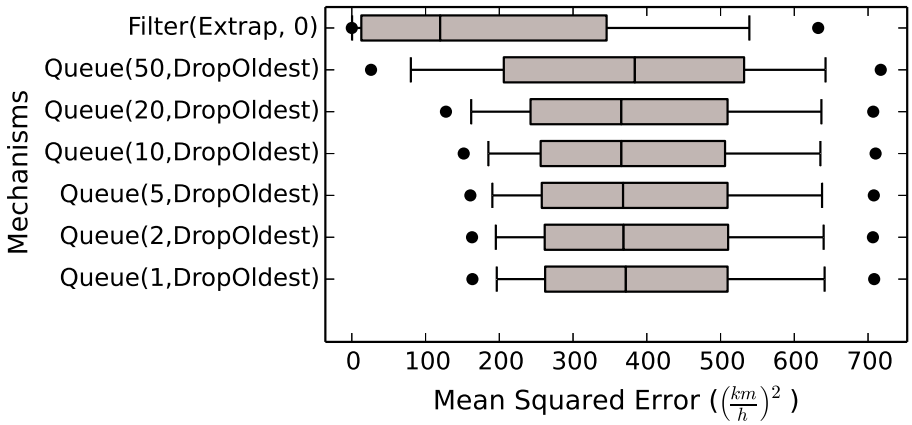


Figure A.22. MSE Comparison for Queue and Filter Mechanisms (Beechwood Data Set)

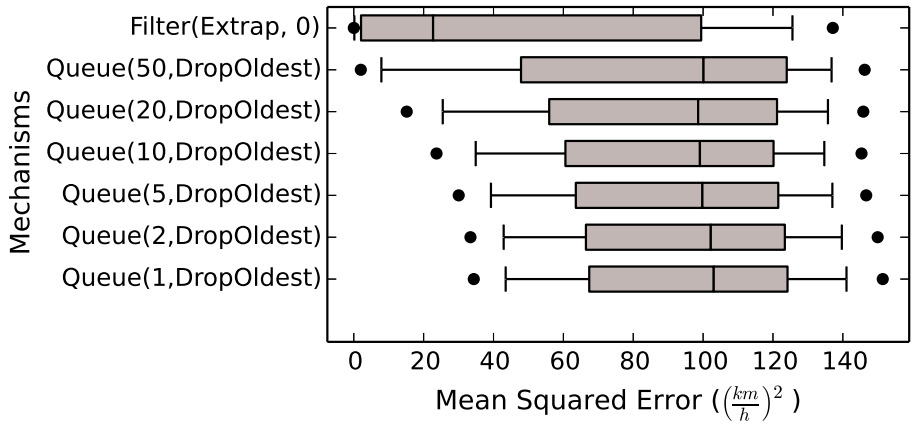


Figure A.23. MSE Comparison for Queue and Filter Mechanisms (Monroeville II Data Set)

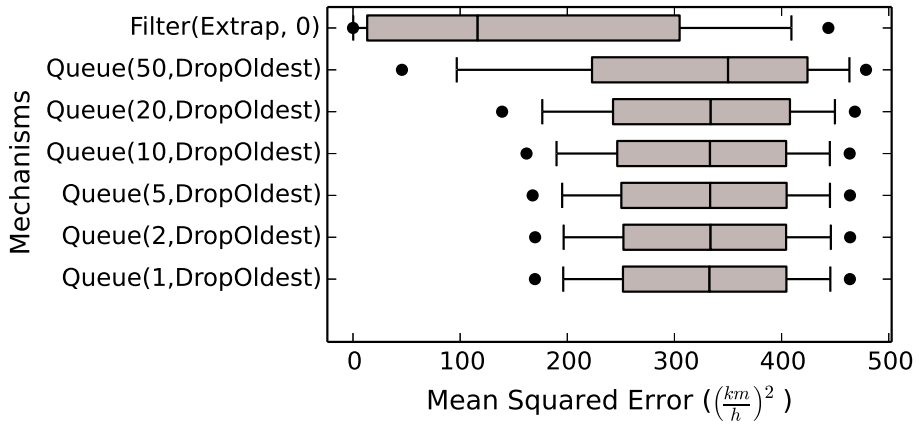


Figure A.24. MSE Comparison for Queue and Filter Mechanisms (Squirrel Hill Data Set)

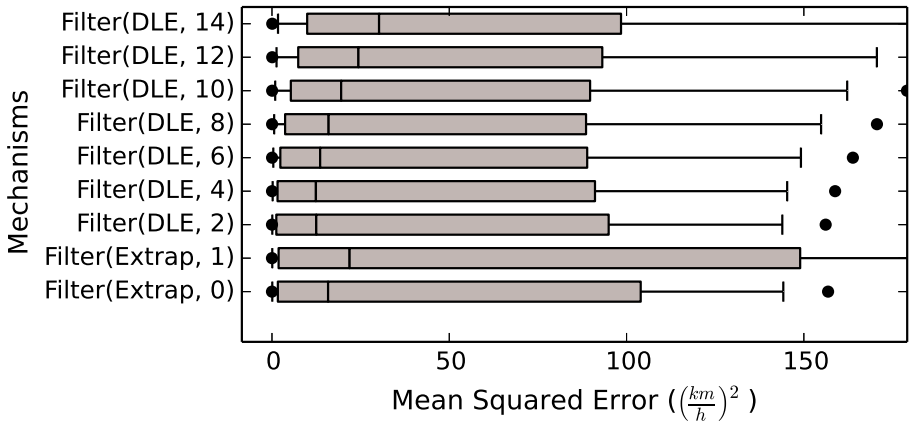


Figure A.25. MSE Comparison for Filter Mechanisms (Monroeville I Data Set)

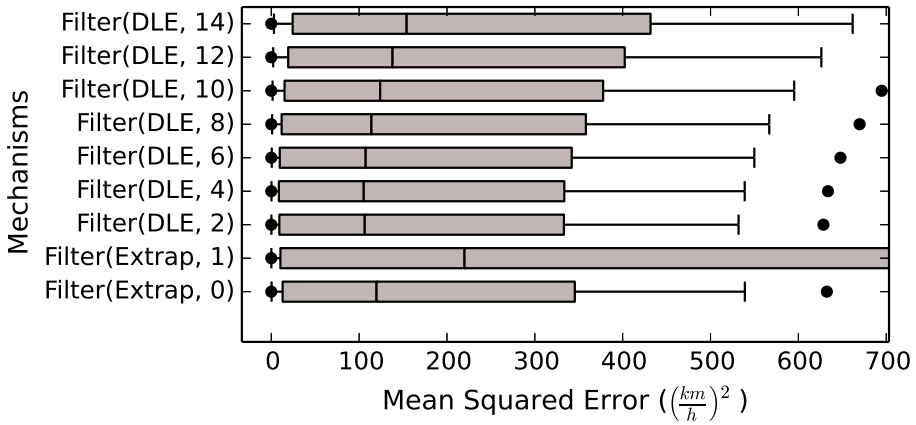


Figure A.26. MSE Comparison for Filter Mechanisms (Beechwood Data Set)

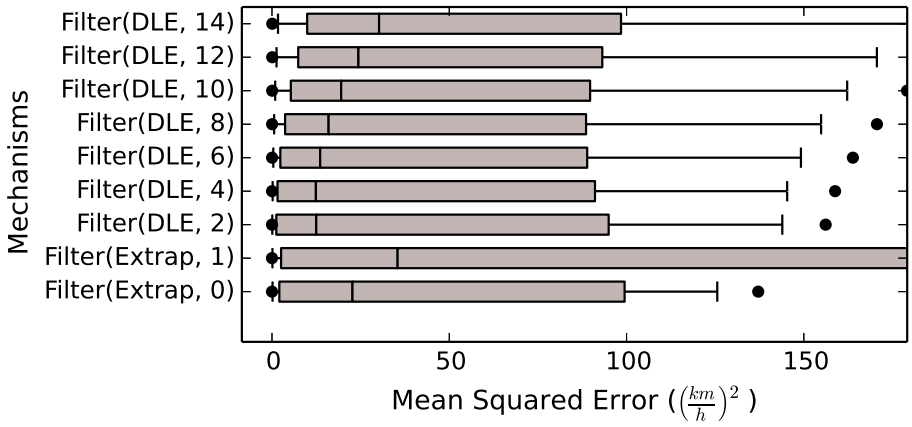


Figure A.27. MSE Comparison for Filter Mechanisms (Monroeville II Data Set)

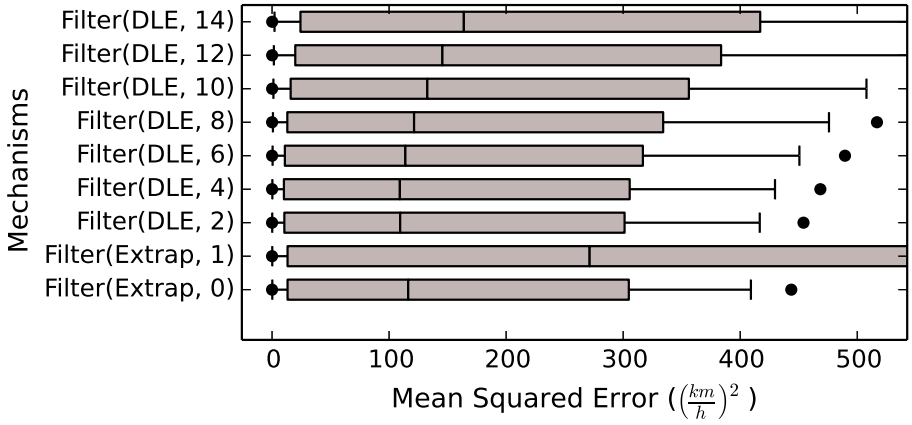


Figure A.28. MSE Comparison for Filter Mechanisms (Squirrel Hill Data Set)

Appendix B

Traffic Case Study Evaluations

This appendix provides the extended results for the evaluation of queue and filter mechanisms using the simulation framework described in Chapter 7. The results of the experiment are presented as a table of the mean values for each of four metrics: flow, motionless ratio, Energy-model Fuel Consumption and Normalized Fuel Ratio. For a complete discussion of the metrics used, see Section 7.2. A 95% confidence interval for each mean value (obtained using bootstrap analysis [53]) is also given in the table.

B.1. Internet Link Congestion Scenario

Table B.1 shows the results from the Internet Link Congestion Scenario described in Section 7.5.1.

Table B.1: Simulation Results for Internet Link Congestion Scenario

Mean Internet Bit Rate (bps)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
13800	15	Queue(1, DropOldest)	4.997 *+5.3e-04* *-5.0e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.0830 *+5.2e-04* *-5.6e-04*	1.8104 *+8.9e-04* *-9.4e-04*
		Queue(10, DropOldest)	4.997 *+5.3e-04* *-5.0e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.0830 *+5.2e-04* *-5.6e-04*	1.8104 *+8.9e-04* *-9.4e-04*
		Queue(20, DropOldest)	4.997 *+5.3e-04* *-5.0e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.0830 *+5.2e-04* *-5.6e-04*	1.8104 *+8.9e-04* *-9.4e-04*
		Queue(50, DropOldest)	4.997 *+5.3e-04* *-5.0e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.0830 *+5.2e-04* *-5.6e-04*	1.8104 *+8.9e-04* *-9.4e-04*
		Filter(Extrap,0)	**4.999** ***+6.1e-04*** ***-6.6e-04***	**0.00000** ***+1.0e-04*** ***-0.0e+00***	1.0814 *+5.8e-04* *-5.3e-04*	1.8075 *+9.3e-04* *-9.1e-04*
		Filter(Extrap,1)	**4.999** ***+6.2e-04*** ***-6.8e-04***	**0.00000** ***+1.0e-04*** ***-0.0e+00***	1.0820 *+5.5e-04* *-5.3e-04*	1.8087 *+9.3e-04* *-9.0e-04*
		Filter(DLE,4)	**4.999** ***+6.1e-04*** ***-6.8e-04***	**0.00000** ***+1.0e-04*** ***-0.0e+00***	1.0820 *+5.5e-04* *-5.3e-04*	1.8087 *+9.2e-04* *-9.0e-04*
		Filter(DLE,8)	**4.999** ***+6.3e-04*** ***-6.8e-04***	**0.00000** ***+1.0e-04*** ***-0.0e+00***	1.0820 *+5.6e-04* *-5.5e-04*	1.8087 *+9.1e-04* *-9.4e-04*

continued on next page

Table B.1: Simulation Results for Internet Link Congestion Scenario – continued

Mean Internet Bit Rate (bps)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
27600	15	Queue(1, DropOldest)	4.997 *+5.2e-04* *-5.0e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.0840 *+5.0e-04* *-5.2e-04*	1.8121 *+8.4e-04* *-8.7e-04*
		Queue(10, DropOldest)	4.997 *+5.2e-04* *-5.0e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.0840 *+5.0e-04* *-5.2e-04*	1.8121 *+8.4e-04* *-8.7e-04*
		Queue(20, DropOldest)	4.997 *+5.2e-04* *-5.0e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.0840 *+5.0e-04* *-5.2e-04*	1.8121 *+8.4e-04* *-8.7e-04*
		Queue(50, DropOldest)	4.997 *+5.2e-04* *-5.0e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.0840 *+5.0e-04* *-5.2e-04*	1.8121 *+8.4e-04* *-8.7e-04*
		Filter(Extrap,0)	**4.998** ***+6.0e-04*** ***-6.7e-04***	**0.00000** ***+1.0e-04*** ***-0.0e+00***	1.0842 *+4.8e-04* *-5.4e-04*	1.8124 *+8.6e-04* *-9.2e-04*
		Filter(Extrap,1)	**4.998** ***+5.8e-04*** ***-6.6e-04***	**0.00000** ***+1.0e-04*** ***-0.0e+00***	1.0849 *+5.1e-04* *-5.3e-04*	1.8137 *+8.4e-04* *-9.2e-04*
		Filter(DLE,4)	**4.998** ***+6.1e-04*** ***-6.7e-04***	**0.00000** ***+1.0e-04*** ***-0.0e+00***	1.0852 *+5.3e-04* *-5.5e-04*	1.8143 *+8.4e-04* *-8.9e-04*
		Filter(DLE,8)	**4.998** ***+5.9e-04*** ***-6.7e-04***	**0.00000** ***+1.0e-04*** ***-0.0e+00***	1.0849 *+5.5e-04* *-5.1e-04*	1.8139 *+8.7e-04* *-8.6e-04*

continued on next page

Table B.1: Simulation Results for Internet Link Congestion Scenario – continued

Mean Internet Bit Rate (bps)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
69000	15	Queue(1, DropOldest)	4.997 *+5.4e-04* *-5.9e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.1241 *+6.1e-04* *-6.7e-04*	1.8772 *+9.9e-04* *-1.1e-03*
		Queue(10, DropOldest)	4.997 *+5.4e-04* *-5.9e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.1241 *+6.1e-04* *-6.7e-04*	1.8772 *+9.9e-04* *-1.1e-03*
		Queue(20, DropOldest)	4.997 *+5.4e-04* *-5.9e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.1241 *+6.1e-04* *-6.7e-04*	1.8772 *+9.9e-04* *-1.1e-03*
		Queue(50, DropOldest)	4.997 *+5.4e-04* *-5.9e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.1241 *+6.1e-04* *-6.7e-04*	1.8772 *+9.9e-04* *-1.1e-03*
		Filter(Extrap,0)	**4.999** ***+6.7e-04*** ***-8.1e-04***	**0.00000** ***+1.0e-04*** ***-0.0e+00***	1.1253 *+5.5e-04* *-6.3e-04*	1.8791 *+9.1e-04* *-1.1e-03*
		Filter(Extrap,1)	**4.999** ***+6.5e-04*** ***-7.8e-04***	**0.00000** ***+1.0e-04*** ***-0.0e+00***	1.1309 *+5.5e-04* *-5.7e-04*	1.8903 *+9.4e-04* *-9.6e-04*
		Filter(DLE,4)	**4.999** ***+6.4e-04*** ***-7.8e-04***	**0.00000** ***+1.0e-04*** ***-0.0e+00***	1.1311 *+5.7e-04* *-5.7e-04*	1.8905 *+9.4e-04* *-9.5e-04*
		Filter(DLE,8)	**4.999** ***+6.6e-04*** ***-7.8e-04***	**0.00000** ***+1.0e-04*** ***-0.0e+00***	1.1315 *+5.5e-04* *-6.4e-04*	1.8914 *+8.9e-04* *-1.1e-03*

continued on next page

Table B.1: Simulation Results for Internet Link Congestion Scenario – continued

Mean Internet Bit Rate (bps)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
110400	15	Queue(1, DropOldest)	4.997 *+6.0e-04* *-6.8e-04*	**0.00001** ***+5.8e-05*** ***-1.4e-05***	1.1580 *+7.1e-04* *-7.3e-04*	1.9316 *+1.1e-03* *-1.2e-03*
		Queue(10, DropOldest)	4.997 *+6.0e-04* *-6.8e-04*	**0.00001** ***+5.8e-05*** ***-1.4e-05***	1.1580 *+7.1e-04* *-7.3e-04*	1.9316 *+1.1e-03* *-1.2e-03*
		Queue(20, DropOldest)	4.997 *+6.0e-04* *-6.8e-04*	**0.00001** ***+5.8e-05*** ***-1.4e-05***	1.1580 *+7.1e-04* *-7.3e-04*	1.9316 *+1.1e-03* *-1.2e-03*
		Queue(50, DropOldest)	4.997 *+6.0e-04* *-6.8e-04*	**0.00001** ***+5.8e-05*** ***-1.4e-05***	1.1580 *+7.1e-04* *-7.3e-04*	1.9316 *+1.1e-03* *-1.2e-03*
		Filter(Extrap,0)	**4.999** ***+7.2e-04*** ***-8.5e-04***	**0.00000** ***+1.2e-04*** ***-0.0e+00***	1.1581 *+6.4e-04* *-7.2e-04*	1.9312 *+1.1e-03* *-1.2e-03*
		Filter(Extrap,1)	**4.999** ***+8.3e-04*** ***-1.0e-03***	**0.00013** ***+1.2e-04*** ***-1.0e-04***	1.1805 *+6.8e-04* *-5.9e-04*	1.9772 *+1.2e-03* *-1.0e-03*
		Filter(DLE,4)	**4.999** ***+7.7e-04*** ***-9.0e-04***	**0.00000** ***+1.2e-04*** ***-0.0e+00***	1.1757 *+6.0e-04* *-6.7e-04*	1.9662 *+1.0e-03* *-1.1e-03*
		Filter(DLE,8)	**4.999** ***+8.1e-04*** ***-1.2e-03***	**0.00000** ***+1.2e-04*** ***-7.5e-07***	1.1769 *+6.3e-04* *-6.6e-04*	1.9698 *+1.0e-03* *-1.1e-03*

continued on next page

Table B.1: Simulation Results for Internet Link Congestion Scenario – continued

Mean Internet Bit Rate (bps)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
13800	30	Queue(1, DropOldest)	**9.985** ***+1.3e-03*** ***-1.3e-03***	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.0793 *+3.7e-04* *-3.6e-04*	1.8055 *+6.0e-04* *-6.6e-04*
		Queue(10, DropOldest)	**9.985** ***+1.3e-03*** ***-1.3e-03***	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.0793 *+3.7e-04* *-3.6e-04*	1.8055 *+6.0e-04* *-6.6e-04*
		Queue(20, DropOldest)	**9.985** ***+1.3e-03*** ***-1.3e-03***	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.0793 *+3.7e-04* *-3.6e-04*	1.8055 *+6.0e-04* *-6.6e-04*
		Queue(50, DropOldest)	**9.985** ***+1.3e-03*** ***-1.3e-03***	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.0793 *+3.7e-04* *-3.6e-04*	1.8055 *+6.0e-04* *-6.6e-04*
		Filter(Extrap,0)	**9.987** ***+1.5e-03*** ***-1.5e-03***	**0.00000** ***+1.1e-04*** ***-0.0e+00***	1.0790 *+3.9e-04* *-3.8e-04*	1.8045 *+6.6e-04* *-6.5e-04*
		Filter(Extrap,1)	**9.987** ***+1.5e-03*** ***-1.5e-03***	**0.00000** ***+1.1e-04*** ***-0.0e+00***	1.0836 *+3.9e-04* *-4.0e-04*	1.8143 *+6.5e-04* *-7.1e-04*
		Filter(DLE,4)	**9.987** ***+1.5e-03*** ***-1.5e-03***	**0.00000** ***+1.1e-04*** ***-0.0e+00***	1.0842 *+3.8e-04* *-4.1e-04*	1.8156 *+6.4e-04* *-6.9e-04*
		Filter(DLE,8)	**9.987** ***+1.5e-03*** ***-1.5e-03***	**0.00000** ***+1.1e-04*** ***-0.0e+00***	1.0852 *+3.8e-04* *-4.0e-04*	1.8175 *+6.3e-04* *-7.1e-04*

continued on next page

Table B.1: Simulation Results for Internet Link Congestion Scenario – continued

Mean Internet Bit Rate (bps)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
27600	30	Queue(1, DropOldest)	**9.985** ***+1.4e-03*** ***-1.3e-03***	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.0819 *+3.5e-04* *-3.4e-04*	1.8097 *+5.4e-04* *-5.6e-04*
		Queue(10, DropOldest)	**9.985** ***+1.4e-03*** ***-1.3e-03***	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.0819 *+3.5e-04* *-3.4e-04*	1.8097 *+5.4e-04* *-5.6e-04*
		Queue(20, DropOldest)	**9.985** ***+1.4e-03*** ***-1.3e-03***	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.0819 *+3.5e-04* *-3.4e-04*	1.8097 *+5.4e-04* *-5.6e-04*
		Queue(50, DropOldest)	**9.985** ***+1.4e-03*** ***-1.3e-03***	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.0819 *+3.5e-04* *-3.4e-04*	1.8097 *+5.4e-04* *-5.6e-04*
		Filter(Extrap,0)	**9.987** ***+1.4e-03*** ***-1.6e-03***	**0.00000** ***+1.1e-04*** ***-0.0e+00***	1.0814 *+3.9e-04* *-3.8e-04*	1.8084 *+6.2e-04* *-6.1e-04*
		Filter(Extrap,1)	**9.987** ***+1.3e-03*** ***-1.6e-03***	**0.00000** ***+1.1e-04*** ***-0.0e+00***	1.0882 *+3.6e-04* *-4.1e-04*	1.8222 *+6.2e-04* *-6.6e-04*
		Filter(DLE,4)	**9.987** ***+1.5e-03*** ***-1.6e-03***	**0.00000** ***+1.1e-04*** ***-0.0e+00***	1.0892 *+4.3e-04* *-3.5e-04*	1.8239 *+7.4e-04* *-6.0e-04*
		Filter(DLE,8)	**9.987** ***+1.5e-03*** ***-1.6e-03***	**0.00000** ***+1.1e-04*** ***-0.0e+00***	1.0897 *+3.9e-04* *-4.1e-04*	1.8251 *+6.4e-04* *-6.7e-04*

continued on next page

Table B.1: Simulation Results for Internet Link Congestion Scenario – continued

Mean Internet Bit Rate (bps)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
69000	30	Queue(1, DropOldest)	9.984 *+1.3e-03* *-1.3e-03*	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.1119 *+4.0e-04* *-4.1e-04*	1.8587 *+6.7e-04* *-7.0e-04*
		Queue(10, DropOldest)	9.984 *+1.3e-03* *-1.3e-03*	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.1119 *+4.0e-04* *-4.1e-04*	1.8587 *+6.7e-04* *-7.0e-04*
		Queue(20, DropOldest)	9.984 *+1.3e-03* *-1.3e-03*	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.1119 *+4.0e-04* *-4.1e-04*	1.8587 *+6.7e-04* *-7.0e-04*
		Queue(50, DropOldest)	9.984 *+1.3e-03* *-1.3e-03*	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.1119 *+4.0e-04* *-4.1e-04*	1.8587 *+6.7e-04* *-7.0e-04*
		Filter(Extrap,0)	**9.988** ***+1.5e-03*** ***-1.7e-03***	**0.00000** ***+1.2e-04*** ***-0.0e+00***	1.1110 *+4.1e-04* *-3.9e-04*	1.8567 *+6.7e-04* *-6.6e-04*
		Filter(Extrap,1)	**9.988** ***+1.6e-03*** ***-1.7e-03***	**0.00001** ***+1.2e-04*** ***-7.1e-06***	1.1274 *+4.1e-04* *-4.1e-04*	1.8897 *+6.9e-04* *-7.2e-04*
		Filter(DLE,4)	**9.988** ***+1.5e-03*** ***-1.7e-03***	**0.00000** ***+1.2e-04*** ***-0.0e+00***	1.1247 *+4.8e-04* *-4.3e-04*	1.8846 *+7.9e-04* *-7.3e-04*
		Filter(DLE,8)	**9.988** ***+1.6e-03*** ***-1.5e-03***	**0.00000** ***+1.2e-04*** ***-0.0e+00***	1.1278 *+3.7e-04* *-4.0e-04*	1.8907 *+6.3e-04* *-6.5e-04*

continued on next page

Table B.1: Simulation Results for Internet Link Congestion Scenario – continued

Mean Internet Bit Rate (bps)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
110400	30	Queue(1, DropOldest)	9.984 *+1.5e-03* *-1.5e-03*	**0.00000** ***+7.0e-05*** ***-2.2e-06***	1.1438 *+4.5e-04* *-4.4e-04*	1.9101 *+7.2e-04* *-7.0e-04*
		Queue(10, DropOldest)	9.984 *+1.5e-03* *-1.5e-03*	**0.00000** ***+7.0e-05*** ***-2.2e-06***	1.1438 *+4.5e-04* *-4.4e-04*	1.9101 *+7.2e-04* *-7.0e-04*
		Queue(20, DropOldest)	9.984 *+1.5e-03* *-1.5e-03*	**0.00000** ***+7.0e-05*** ***-2.2e-06***	1.1438 *+4.5e-04* *-4.4e-04*	1.9101 *+7.2e-04* *-7.0e-04*
		Queue(50, DropOldest)	9.984 *+1.5e-03* *-1.5e-03*	**0.00000** ***+7.0e-05*** ***-2.2e-06***	1.1438 *+4.5e-04* *-4.4e-04*	1.9101 *+7.2e-04* *-7.0e-04*
		Filter(Extrap,0)	**9.988** ***+1.5e-03*** ***-1.8e-03***	**0.00000** ***+1.3e-04*** ***-3.6e-06***	1.1451 *+4.6e-04* *-4.7e-04*	1.9115 *+7.5e-04* *-7.5e-04*
		Filter(Extrap,1)	**9.988** ***+2.1e-03*** ***-2.5e-03***	0.00026 *+1.3e-04* *-1.0e-04*	1.1861 *+5.2e-04* *-4.5e-04*	1.9978 *+9.2e-04* *-7.9e-04*
		Filter(DLE,4)	**9.988** ***+1.7e-03*** ***-2.1e-03***	**0.00000** ***+1.3e-04*** ***-9.3e-08***	1.1770 *+5.2e-04* *-5.1e-04*	1.9752 *+9.0e-04* *-8.2e-04*
		Filter(DLE,8)	**9.988** ***+1.8e-03*** ***-2.3e-03***	**0.00001** ***+1.2e-04*** ***-6.1e-06***	1.1844 *+4.8e-04* *-5.3e-04*	1.9929 *+7.9e-04* *-8.7e-04*

B.2. Noisy Wireless Network Scenario

Table B.2 shows the results from the Noisy Wireless Network Scenario described in Section 7.5.2.

Table B.2: Simulation Results for Noise Wireless Network Scenario

Packet RX Power Threshold (dB)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
-85	15	Queue(1, DropOldest)	4.997 *+4.9e-04* *-5.0e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.0799 *+4.9e-04* *-5.3e-04*	1.8053 *+8.6e-04* *-8.8e-04*
		Queue(10, DropOldest)	4.997 *+4.9e-04* *-5.0e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.0799 *+4.9e-04* *-5.3e-04*	1.8053 *+8.6e-04* *-8.8e-04*
		Queue(20, DropOldest)	4.997 *+4.9e-04* *-5.0e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.0799 *+4.9e-04* *-5.3e-04*	1.8053 *+8.6e-04* *-8.8e-04*
		Queue(50, DropOldest)	4.997 *+4.9e-04* *-5.0e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.0799 *+4.9e-04* *-5.3e-04*	1.8053 *+8.6e-04* *-8.8e-04*
		Filter(Extrap,0)	**4.999** ***+5.9e-04*** ***-6.9e-04***	**0.00000** ***+1.0e-04*** ***-0.0e+00***	1.0794 *+5.0e-04* *-5.2e-04*	1.8042 *+8.7e-04* *-9.0e-04*
		Filter(Extrap,1)	**4.999** ***+5.9e-04*** ***-6.9e-04***	**0.00000** ***+1.0e-04*** ***-0.0e+00***	1.0798 *+5.0e-04* *-5.4e-04*	1.8050 *+8.4e-04* *-9.1e-04*
		Filter(DLE,4)	**4.999** ***+5.9e-04*** ***-6.9e-04***	**0.00000** ***+1.0e-04*** ***-0.0e+00***	1.0797 *+5.0e-04* *-5.5e-04*	1.8050 *+8.5e-04* *-9.1e-04*
		Filter(DLE,8)	**4.999** ***+5.9e-04*** ***-6.8e-04***	**0.00000** ***+1.0e-04*** ***-0.0e+00***	1.0798 *+5.1e-04* *-5.2e-04*	1.8051 *+8.9e-04* *-8.8e-04*

continued on next page

Table B.2: Simulation Results for Noise Wireless Network Scenario – continued

Packet RX Power Threshold (dB)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
-75	15	Queue(1, DropOldest)	4.997 *+4.7e-04* *-5.1e-04*	**0.00000** ***+6.2e-05*** ***-9.3e-07***	0.9662 *+1.3e-04* *-1.3e-04*	1.6045 *+2.6e-04* *-2.7e-04*
		Queue(10, DropOldest)	4.997 *+4.7e-04* *-5.1e-04*	**0.00000** ***+6.2e-05*** ***-9.3e-07***	0.9662 *+1.3e-04* *-1.3e-04*	1.6045 *+2.6e-04* *-2.7e-04*
		Queue(20, DropOldest)	4.997 *+4.7e-04* *-5.1e-04*	**0.00000** ***+6.2e-05*** ***-9.3e-07***	0.9662 *+1.3e-04* *-1.3e-04*	1.6045 *+2.6e-04* *-2.7e-04*
		Queue(50, DropOldest)	4.997 *+4.7e-04* *-5.1e-04*	**0.00000** ***+6.2e-05*** ***-9.3e-07***	0.9662 *+1.3e-04* *-1.3e-04*	1.6045 *+2.6e-04* *-2.7e-04*
		Filter(Extrap,0)	**4.999** ***+5.7e-04*** ***-6.7e-04***	**0.00000** ***+1.0e-04*** ***-9.3e-07***	0.9657 *+1.1e-04* *-1.2e-04*	1.6033 *+2.5e-04* *-2.6e-04*
		Filter(Extrap,1)	4.995 *+7.2e-04* *-7.9e-04*	0.00074 *+1.3e-04* *-1.2e-04*	0.9825 *+2.3e-04* *-2.6e-04*	1.6396 *+4.5e-04* *-5.0e-04*
		Filter(DLE,4)	**4.998** ***+6.6e-04*** ***-7.2e-04***	**0.00004** ***+9.9e-05*** ***-3.9e-05***	0.9967 *+2.9e-04* *-3.2e-04*	1.6681 *+5.2e-04* *-5.9e-04*
		Filter(DLE,8)	**4.998** ***+6.4e-04*** ***-7.2e-04***	**0.00006** ***+1.0e-04*** ***-5.6e-05***	1.0021 *+3.0e-04* *-3.2e-04*	1.6809 *+5.6e-04* *-6.0e-04*

continued on next page

Table B.2: Simulation Results for Noise Wireless Network Scenario – continued

Packet RX Power Threshold (dB)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
-65	15	Queue(1, DropOldest)	**4.996** ***+1.7e-03*** ***-2.1e-03***	0.00492 *+1.8e-04* *-1.8e-04*	1.0149 *+3.0e-04* *-3.1e-04*	1.7478 *+6.7e-04* *-7.3e-04*
		Queue(10, DropOldest)	**4.996** ***+1.7e-03*** ***-2.1e-03***	0.00492 *+1.8e-04* *-1.8e-04*	1.0149 *+3.0e-04* *-3.1e-04*	1.7478 *+6.7e-04* *-7.3e-04*
		Queue(20, DropOldest)	**4.996** ***+1.7e-03*** ***-2.1e-03***	0.00492 *+1.8e-04* *-1.8e-04*	1.0149 *+3.0e-04* *-3.1e-04*	1.7478 *+6.7e-04* *-7.3e-04*
		Queue(50, DropOldest)	**4.996** ***+1.7e-03*** ***-2.1e-03***	0.00492 *+1.8e-04* *-1.8e-04*	1.0149 *+3.0e-04* *-3.1e-04*	1.7478 *+6.7e-04* *-7.3e-04*
		Filter(Extrap,0)	**4.997** ***+1.7e-03*** ***-1.8e-03***	**0.00447** ***+2.0e-04*** ***-2.0e-04***	1.0133 *+2.7e-04* *-3.1e-04*	1.7422 *+6.0e-04* *-7.4e-04*
		Filter(Extrap,1)	0.429 *+4.6e-03* *-4.9e-03*	0.91146 *+9.8e-04* *-9.8e-04*	0.4919 *+1.6e-04* *-1.8e-04*	1.0604 *+2.7e-04* *-3.1e-04*
		Filter(DLE,4)	4.845 *+2.4e-03* *-2.6e-03*	0.01134 *+3.0e-04* *-2.5e-04*	1.0249 *+3.6e-04* *-3.8e-04*	1.7838 *+8.7e-04* *-7.6e-04*
		Filter(DLE,8)	4.513 *+4.2e-03* *-4.1e-03*	0.02724 *+5.3e-04* *-4.1e-04*	0.9941 *+4.0e-04* *-4.0e-04*	1.7643 *+8.2e-04* *-9.0e-04*

continued on next page

Table B.2: Simulation Results for Noise Wireless Network Scenario – continued

Packet RX Power Threshold (dB)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
-55	15	Queue(1, DropOldest)	4.977 *+1.8e-03* *-1.8e-03*	**0.00624** ***+1.9e-04*** ***-1.8e-04***	0.9955 *+2.3e-04* *-2.8e-04*	1.7065 *+6.4e-04* *-7.1e-04*
		Queue(10, DropOldest)	4.977 *+1.8e-03* *-1.8e-03*	**0.00624** ***+1.9e-04*** ***-1.8e-04***	0.9955 *+2.3e-04* *-2.8e-04*	1.7065 *+6.4e-04* *-7.1e-04*
		Queue(20, DropOldest)	4.977 *+1.8e-03* *-1.8e-03*	**0.00624** ***+1.9e-04*** ***-1.8e-04***	0.9955 *+2.3e-04* *-2.8e-04*	1.7065 *+6.4e-04* *-7.1e-04*
		Queue(50, DropOldest)	4.977 *+1.8e-03* *-1.8e-03*	**0.00624** ***+1.9e-04*** ***-1.8e-04***	0.9955 *+2.3e-04* *-2.8e-04*	1.7065 *+6.4e-04* *-7.1e-04*
		Filter(Extrap,0)	**5.009** ***+2.0e-03*** ***-2.3e-03***	**0.00638** ***+2.9e-04*** ***-2.6e-04***	1.0131 *+2.9e-04* *-2.8e-04*	1.7591 *+7.4e-04* *-7.9e-04*
		Filter(Extrap,1)	0.214 *+3.2e-03* *-3.6e-03*	0.95647 *+7.3e-04* *-6.6e-04*	0.4680 *+1.5e-04* *-1.5e-04*	1.0321 *+2.5e-04* *-3.0e-04*
		Filter(DLE,4)	3.917 *+4.9e-03* *-4.8e-03*	0.10151 *+9.5e-04* *-8.2e-04*	0.8992 *+3.7e-04* *-4.2e-04*	1.6516 *+9.1e-04* *-8.5e-04*
		Filter(DLE,8)	3.182 *+6.3e-03* *-5.6e-03*	0.10796 *+9.1e-04* *-9.3e-04*	0.8201 *+3.9e-04* *-3.9e-04*	1.5469 *+8.4e-04* *-8.8e-04*

continued on next page

Table B.2: Simulation Results for Noise Wireless Network Scenario – continued

Packet RX Power Threshold (dB)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
-85	30	Queue(1, DropOldest)	**9.985** ***+1.3e-03*** ***-1.3e-03***	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.0767 *+3.7e-04* *-3.5e-04*	1.8011 *+6.4e-04* *-5.9e-04*
		Queue(10, DropOldest)	**9.985** ***+1.3e-03*** ***-1.3e-03***	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.0767 *+3.7e-04* *-3.5e-04*	1.8011 *+6.4e-04* *-5.9e-04*
		Queue(20, DropOldest)	**9.985** ***+1.3e-03*** ***-1.3e-03***	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.0767 *+3.7e-04* *-3.5e-04*	1.8011 *+6.4e-04* *-5.9e-04*
		Queue(50, DropOldest)	**9.985** ***+1.3e-03*** ***-1.3e-03***	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.0767 *+3.7e-04* *-3.5e-04*	1.8011 *+6.4e-04* *-5.9e-04*
		Filter(Extrap,0)	**9.987** ***+1.4e-03*** ***-1.5e-03***	**0.00000** ***+1.1e-04*** ***-0.0e+00***	1.0757 *+3.8e-04* *-3.6e-04*	1.7991 *+6.7e-04* *-6.2e-04*
		Filter(Extrap,1)	**9.987** ***+1.5e-03*** ***-1.5e-03***	**0.00000** ***+1.1e-04*** ***-2.8e-07***	1.0937 *+4.2e-04* *-4.1e-04*	1.8355 *+6.8e-04* *-7.1e-04*
		Filter(DLE,4)	**9.987** ***+1.5e-03*** ***-1.6e-03***	**0.00000** ***+1.1e-04*** ***-0.0e+00***	1.0945 *+3.7e-04* *-3.8e-04*	1.8389 *+6.7e-04* *-6.7e-04*
		Filter(DLE,8)	**9.987** ***+1.7e-03*** ***-1.6e-03***	**0.00000** ***+1.1e-04*** ***-5.6e-07***	1.0982 *+3.8e-04* *-3.8e-04*	1.8477 *+6.6e-04* *-6.7e-04*

continued on next page

Table B.2: Simulation Results for Noise Wireless Network Scenario – continued

Packet RX Power Threshold (dB)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
-75	30	Queue(1, DropOldest)	**9.984** ***+1.6e-03*** ***-1.7e-03***	**0.00004** ***+6.9e-05*** ***-4.4e-05***	0.9746 *+1.3e-04* *-1.2e-04*	1.6259 *+2.7e-04* *-2.3e-04*
		Queue(10, DropOldest)	**9.984** ***+1.6e-03*** ***-1.7e-03***	**0.00004** ***+6.9e-05*** ***-4.4e-05***	0.9746 *+1.3e-04* *-1.2e-04*	1.6259 *+2.7e-04* *-2.3e-04*
		Queue(20, DropOldest)	**9.984** ***+1.6e-03*** ***-1.7e-03***	**0.00004** ***+6.9e-05*** ***-4.4e-05***	0.9746 *+1.3e-04* *-1.2e-04*	1.6259 *+2.7e-04* *-2.3e-04*
		Queue(50, DropOldest)	**9.984** ***+1.6e-03*** ***-1.7e-03***	**0.00004** ***+6.9e-05*** ***-4.4e-05***	0.9746 *+1.3e-04* *-1.2e-04*	1.6259 *+2.7e-04* *-2.3e-04*
		Filter(Extrap,0)	**9.987** ***+1.6e-03*** ***-1.6e-03***	**0.00007** ***+1.1e-04*** ***-6.8e-05***	0.9776 *+1.5e-04* *-1.4e-04*	1.6325 *+3.0e-04* *-2.8e-04*
		Filter(Extrap,1)	9.206 *+9.1e-03* *-9.3e-03*	0.07760 *+9.3e-04* *-8.6e-04*	0.9425 *+1.8e-04* *-2.0e-04*	1.5983 *+3.4e-04* *-3.8e-04*
		Filter(DLE,4)	**9.987** ***+1.5e-03*** ***-2.0e-03***	**0.00009** ***+1.1e-04*** ***-9.3e-05***	1.0019 *+2.2e-04* *-2.2e-04*	1.6829 *+4.1e-04* *-4.0e-04*
		Filter(DLE,8)	**9.985** ***+2.0e-03*** ***-1.8e-03***	**0.00016** ***+1.1e-04*** ***-9.5e-05***	1.0221 *+2.7e-04* *-2.6e-04*	1.7249 *+4.8e-04* *-5.0e-04*

continued on next page

Table B.2: Simulation Results for Noise Wireless Network Scenario – continued

Packet RX Power Threshold (dB)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
-65	30	Queue(1, DropOldest)	9.899 *+2.8e-03* *-2.8e-03*	0.00364 *+1.6e-04* *-1.4e-04*	0.9840 *+1.6e-04* *-1.9e-04*	1.6574 *+3.3e-04* *-3.9e-04*
		Queue(10, DropOldest)	9.899 *+2.8e-03* *-2.8e-03*	0.00364 *+1.6e-04* *-1.4e-04*	0.9840 *+1.6e-04* *-1.9e-04*	1.6574 *+3.3e-04* *-3.9e-04*
		Queue(20, DropOldest)	9.899 *+2.8e-03* *-2.8e-03*	0.00364 *+1.6e-04* *-1.4e-04*	0.9840 *+1.6e-04* *-1.9e-04*	1.6574 *+3.3e-04* *-3.9e-04*
		Queue(50, DropOldest)	9.899 *+2.8e-03* *-2.8e-03*	0.00364 *+1.6e-04* *-1.4e-04*	0.9840 *+1.6e-04* *-1.9e-04*	1.6574 *+3.3e-04* *-3.9e-04*
		Filter(Extrap,0)	**9.982** ***+2.9e-03*** ***-3.3e-03***	**0.00183** ***+2.8e-04*** ***-2.4e-04***	0.9899 *+1.8e-04* *-1.8e-04*	1.6614 *+3.3e-04* *-3.6e-04*
		Filter(Extrap,1)	4.261 *+1.5e-02* *-1.7e-02*	0.56057 *+1.5e-03* *-1.5e-03*	0.7022 *+2.8e-04* *-2.4e-04*	1.3481 *+4.9e-04* *-4.3e-04*
		Filter(DLE,4)	9.761 *+5.0e-03* *-4.6e-03*	0.00928 *+3.5e-04* *-3.1e-04*	1.0428 *+3.0e-04* *-3.1e-04*	1.8051 *+6.6e-04* *-6.4e-04*
		Filter(DLE,8)	9.391 *+7.6e-03* *-7.1e-03*	0.02597 *+5.1e-04* *-5.5e-04*	1.0335 *+3.1e-04* *-3.5e-04*	1.8162 *+6.3e-04* *-7.4e-04*

continued on next page

Table B.2: Simulation Results for Noise Wireless Network Scenario – continued

Packet RX Power Threshold (dB)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
-55	30	Queue(1, DropOldest)	**9.590** ***+5.7e-03*** ***-5.8e-03***	**0.01863** ***+3.6e-04*** ***-3.8e-04***	0.9715 *+1.7e-04* *-1.8e-04*	1.6676 *+4.1e-04* *-4.3e-04*
		Queue(10, DropOldest)	**9.590** ***+5.7e-03*** ***-5.8e-03***	**0.01863** ***+3.6e-04*** ***-3.8e-04***	0.9715 *+1.7e-04* *-1.8e-04*	1.6676 *+4.1e-04* *-4.3e-04*
		Queue(20, DropOldest)	**9.590** ***+5.7e-03*** ***-5.8e-03***	**0.01863** ***+3.6e-04*** ***-3.8e-04***	0.9715 *+1.7e-04* *-1.8e-04*	1.6676 *+4.1e-04* *-4.3e-04*
		Queue(50, DropOldest)	**9.590** ***+5.7e-03*** ***-5.8e-03***	**0.01863** ***+3.6e-04*** ***-3.8e-04***	0.9715 *+1.7e-04* *-1.8e-04*	1.6676 *+4.1e-04* *-4.3e-04*
		Filter(Extrap,0)	9.390 *+6.8e-03* *-6.9e-03*	0.03010 *+5.4e-04* *-5.3e-04*	0.9616 *+1.8e-04* *-1.9e-04*	1.6611 *+4.4e-04* *-3.9e-04*
		Filter(Extrap,1)	0.677 *+7.8e-03* *-6.6e-03*	0.91705 *+8.1e-04* *-8.5e-04*	0.4887 *+1.4e-04* *-1.4e-04*	1.0722 *+2.7e-04* *-2.7e-04*
		Filter(DLE,4)	4.410 *+1.3e-02* *-1.3e-02*	0.45851 *+1.4e-03* *-1.4e-03*	0.7208 *+2.6e-04* *-2.4e-04*	1.4322 *+5.6e-04* *-5.8e-04*
		Filter(DLE,8)	3.895 *+1.5e-02* *-1.4e-02*	0.44110 *+1.6e-03* *-1.6e-03*	0.6931 *+2.7e-04* *-2.5e-04*	1.3913 *+5.1e-04* *-5.7e-04*

B.3. Background Wireless Traffic Congestion Scenario

Table B.3 shows the results from the Background Wireless Traffic Congestion Scenario described in Section 7.5.3.

Table B.3: Simulation Results for Background Wireless Traffic Scenario

Bgnd. Packet Size (bytes)	Bgnd. Mean Send Period (s)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
1024	0.01	15	Queue(1, DropOldest)	**4.961** ***+7.0e-03*** ***-6.4e-03***	**0.00995** ***+4.8e-04*** ***-5.1e-04***	1.0931 *+8.2e-04* *-8.8e-04*	1.9419 *+1.7e-03* *-2.0e-03*
			Queue(10, DropOldest)	**4.961** ***+7.0e-03*** ***-6.4e-03***	**0.00995** ***+4.8e-04*** ***-5.1e-04***	1.0931 *+8.2e-04* *-8.8e-04*	1.9419 *+1.7e-03* *-2.0e-03*
			Queue(20, DropOldest)	**4.961** ***+7.0e-03*** ***-6.4e-03***	**0.00995** ***+4.8e-04*** ***-5.1e-04***	1.0931 *+8.2e-04* *-8.8e-04*	1.9419 *+1.7e-03* *-2.0e-03*
			Queue(50, DropOldest)	**4.961** ***+7.0e-03*** ***-6.4e-03***	**0.00995** ***+4.8e-04*** ***-5.1e-04***	1.0931 *+8.2e-04* *-8.8e-04*	1.9419 *+1.7e-03* *-2.0e-03*
			Filter(Extrap,0)	4.906 *+6.7e-03* *-6.3e-03*	0.01460 *+6.1e-04* *-6.7e-04*	1.0812 *+9.6e-04* *-9.7e-04*	1.9212 *+2.0e-03* *-2.1e-03*
			Filter(Extrap,1)	4.252 *+1.1e-02* *-1.1e-02*	0.13183 *+1.7e-03* *-1.5e-03*	1.0465 *+1.1e-03* *-9.9e-04*	1.9352 *+2.2e-03* *-2.2e-03*
			Filter(DLE,4)	4.710 *+7.7e-03* *-8.2e-03*	0.04027 *+9.4e-04* *-9.3e-04*	1.1129 *+1.1e-03* *-1.0e-03*	1.9602 *+2.2e-03* *-2.1e-03*
			Filter(DLE,8)	4.246 *+9.9e-03* *-1.1e-02*	0.11127 *+1.6e-03* *-1.6e-03*	1.0478 *+1.0e-03* *-1.0e-03*	1.9091 *+2.0e-03* *-2.1e-03*

continued on next page

Table B.3: Simulation Results for Background Wireless Traffic Scenario – continued

Bgnd. Packet Size (bytes)	Bgnd. Mean Send Period (s)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
1024	0.10	15	Queue(1, DropOldest)	**4.996** ***+5.2e-03*** ***-5.6e-03***	0.00366 *+2.5e-04* *-2.4e-04*	1.1579 *+1.3e-03* *-1.2e-03*	2.0297 *+2.2e-03* *-2.2e-03*
			Queue(10, DropOldest)	**4.996** ***+5.2e-03*** ***-5.6e-03***	0.00366 *+2.5e-04* *-2.4e-04*	1.1579 *+1.3e-03* *-1.2e-03*	2.0297 *+2.2e-03* *-2.2e-03*
			Queue(20, DropOldest)	**4.996** ***+5.2e-03*** ***-5.6e-03***	0.00366 *+2.5e-04* *-2.4e-04*	1.1579 *+1.3e-03* *-1.2e-03*	2.0297 *+2.2e-03* *-2.2e-03*
			Queue(50, DropOldest)	**4.996** ***+5.2e-03*** ***-5.6e-03***	0.00366 *+2.5e-04* *-2.4e-04*	1.1579 *+1.3e-03* *-1.2e-03*	2.0297 *+2.2e-03* *-2.2e-03*
			Filter(Extrap,0)	**4.998** ***+4.8e-03*** ***-5.1e-03***	0.00344 *+2.6e-04* *-2.5e-04*	1.1585 *+1.1e-03* *-1.2e-03*	2.0261 *+2.0e-03* *-2.2e-03*
			Filter(Extrap,1)	**4.993** ***+6.6e-03*** ***-6.3e-03***	0.01535 *+6.1e-04* *-5.8e-04*	1.3445 *+1.6e-03* *-1.6e-03*	2.4249 *+3.0e-03* *-3.0e-03*
			Filter(DLE,4)	**4.997** ***+4.5e-03*** ***-4.3e-03***	**0.00023** ***+2.0e-04*** ***-1.8e-04***	1.3874 *+1.7e-03* *-1.7e-03*	2.4274 *+2.9e-03* *-3.0e-03*
			Filter(DLE,8)	**4.996** ***+6.1e-03*** ***-5.6e-03***	**0.00020** ***+2.1e-04*** ***-1.9e-04***	1.3733 *+1.5e-03* *-1.5e-03*	2.4441 *+2.7e-03* *-2.8e-03*

continued on next page

Table B.3: Simulation Results for Background Wireless Traffic Scenario – continued

Bgnd. Packet Size (bytes)	Bgnd. Mean Send Period (s)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
4096	0.01	15	Queue(1, DropOldest)	4.792 *+8.1e-03* *-7.1e-03*	0.01980 *+6.7e-04* *-6.6e-04*	1.0474 *+7.6e-04* *-8.4e-04*	1.8535 *+1.6e-03* *-1.7e-03*
			Queue(10, DropOldest)	4.792 *+8.1e-03* *-7.1e-03*	0.01980 *+6.7e-04* *-6.6e-04*	1.0474 *+7.6e-04* *-8.4e-04*	1.8535 *+1.6e-03* *-1.7e-03*
			Queue(20, DropOldest)	4.792 *+8.1e-03* *-7.1e-03*	0.01980 *+6.7e-04* *-6.6e-04*	1.0474 *+7.6e-04* *-8.4e-04*	1.8535 *+1.6e-03* *-1.7e-03*
			Queue(50, DropOldest)	4.792 *+8.1e-03* *-7.1e-03*	0.01980 *+6.7e-04* *-6.6e-04*	1.0474 *+7.6e-04* *-8.4e-04*	1.8535 *+1.6e-03* *-1.7e-03*
			Filter(Extrap,0)	**4.951** ***+5.4e-03*** ***-6.4e-03***	**0.00939** ***+5.2e-04*** ***-4.9e-04***	1.0708 *+9.6e-04* *-8.6e-04*	1.8857 *+2.0e-03* *-1.9e-03*
			Filter(Extrap,1)	3.159 *+1.3e-02* *-1.4e-02*	0.31272 *+2.4e-03* *-2.4e-03*	0.8986 *+9.2e-04* *-1.0e-03*	1.7319 *+2.1e-03* *-2.3e-03*
			Filter(DLE,4)	3.784 *+1.2e-02* *-1.2e-02*	0.19567 *+2.0e-03* *-1.8e-03*	0.9544 *+9.2e-04* *-8.9e-04*	1.7410 *+1.8e-03* *-1.8e-03*
			Filter(DLE,8)	2.883 *+1.3e-02* *-1.4e-02*	0.35190 *+2.7e-03* *-2.7e-03*	0.8506 *+9.2e-04* *-8.9e-04*	1.6450 *+2.1e-03* *-1.6e-03*

continued on next page

Table B.3: Simulation Results for Background Wireless Traffic Scenario – continued

Bgnd. Packet Size (bytes)	Bgnd. Mean Send Period (s)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
4096	0.10	15	Queue(1, DropOldest)	**4.986** ***+7.3e-03*** ***-7.4e-03***	0.01288 *+5.8e-04* *-5.6e-04*	1.1259 *+1.1e-03* *-1.0e-03*	2.0213 *+2.2e-03* *-2.2e-03*
			Queue(10, DropOldest)	**4.986** ***+7.3e-03*** ***-7.4e-03***	0.01288 *+5.8e-04* *-5.6e-04*	1.1259 *+1.1e-03* *-1.0e-03*	2.0213 *+2.2e-03* *-2.2e-03*
			Queue(20, DropOldest)	**4.986** ***+7.3e-03*** ***-7.4e-03***	0.01288 *+5.8e-04* *-5.6e-04*	1.1259 *+1.1e-03* *-1.0e-03*	2.0213 *+2.2e-03* *-2.2e-03*
			Queue(50, DropOldest)	**4.986** ***+7.3e-03*** ***-7.4e-03***	0.01288 *+5.8e-04* *-5.6e-04*	1.1259 *+1.1e-03* *-1.0e-03*	2.0213 *+2.2e-03* *-2.2e-03*
			Filter(Extrap,0)	**4.989** ***+7.0e-03*** ***-7.3e-03***	0.01276 *+5.9e-04* *-6.1e-04*	1.1235 *+1.0e-03* *-1.0e-03*	2.0189 *+2.1e-03* *-2.2e-03*
			Filter(Extrap,1)	4.799 *+8.0e-03* *-9.1e-03*	0.04707 *+1.1e-03* *-8.7e-04*	1.1797 *+1.2e-03* *-1.1e-03*	2.1638 *+2.3e-03* *-2.4e-03*
			Filter(DLE,4)	**4.979** ***+6.9e-03*** ***-7.1e-03***	**0.00628** ***+3.8e-04*** ***-3.5e-04***	1.2132 *+1.1e-03* *-1.3e-03*	2.1681 *+2.3e-03* *-2.4e-03*
			Filter(DLE,8)	4.889 *+7.8e-03* *-7.6e-03*	0.01731 *+6.9e-04* *-6.7e-04*	1.2058 *+1.3e-03* *-1.3e-03*	2.1693 *+2.4e-03* *-2.3e-03*

continued on next page

Table B.3: Simulation Results for Background Wireless Traffic Scenario – continued

Bgnd. Packet Size (bytes)	Bgnd. Mean Send Period (s)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
1024	0.01	30	Queue(1, DropOldest)	2.868 *+2.9e-02* *-3.0e-02*	0.11542 *+2.7e-03* *-2.3e-03*	0.5967 *+5.1e-04* *-5.3e-04*	1.2622 *+1.5e-03* *-1.7e-03*
			Queue(10, DropOldest)	2.868 *+2.9e-02* *-3.0e-02*	0.11542 *+2.7e-03* *-2.3e-03*	0.5967 *+5.1e-04* *-5.3e-04*	1.2622 *+1.5e-03* *-1.7e-03*
			Queue(20, DropOldest)	2.868 *+2.9e-02* *-3.0e-02*	0.11542 *+2.7e-03* *-2.3e-03*	0.5967 *+5.1e-04* *-5.3e-04*	1.2622 *+1.5e-03* *-1.7e-03*
			Queue(50, DropOldest)	2.868 *+2.9e-02* *-3.0e-02*	0.11542 *+2.7e-03* *-2.3e-03*	0.5967 *+5.1e-04* *-5.3e-04*	1.2622 *+1.5e-03* *-1.7e-03*
			Filter(Extrap,0)	**4.527** ***+3.8e-02*** ***-3.9e-02***	**0.08846** ***+2.4e-03*** ***-2.1e-03***	0.6923 *+7.4e-04* *-6.4e-04*	1.3539 *+1.5e-03* *-1.4e-03*
			Filter(Extrap,1)	2.115 *+4.7e-02* *-4.3e-02*	0.76613 *+4.0e-03* *-4.5e-03*	0.6110 *+9.5e-04* *-8.1e-04*	1.2974 *+1.9e-03* *-1.9e-03*
			Filter(DLE,4)	3.064 *+5.1e-02* *-4.8e-02*	0.65975 *+4.9e-03* *-5.0e-03*	0.6430 *+8.5e-04* *-8.3e-04*	1.3148 *+1.7e-03* *-1.6e-03*
			Filter(DLE,8)	1.384 *+3.5e-02* *-3.3e-02*	0.82852 *+3.6e-03* *-3.5e-03*	0.5427 *+6.3e-04* *-6.9e-04*	1.1726 *+1.3e-03* *-1.5e-03*

continued on next page

Table B.3: Simulation Results for Background Wireless Traffic Scenario – continued

Bgnd. Packet Size (bytes)	Bgnd. Mean Send Period (s)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
1024	0.10	30	Queue(1, DropOldest)	2.153 *+2.5e-02* *-2.5e-02*	**0.13026** ***+2.7e-03*** ***-2.7e-03***	0.5632 *+5.6e-04* *-5.6e-04*	1.1995 *+1.3e-03* *-1.3e-03*
			Queue(10, DropOldest)	2.153 *+2.5e-02* *-2.5e-02*	**0.13026** ***+2.7e-03*** ***-2.7e-03***	0.5632 *+5.6e-04* *-5.6e-04*	1.1995 *+1.3e-03* *-1.3e-03*
			Queue(20, DropOldest)	2.153 *+2.5e-02* *-2.5e-02*	**0.13026** ***+2.7e-03*** ***-2.7e-03***	0.5632 *+5.6e-04* *-5.6e-04*	1.1995 *+1.3e-03* *-1.3e-03*
			Queue(50, DropOldest)	2.153 *+2.5e-02* *-2.5e-02*	**0.13026** ***+2.7e-03*** ***-2.7e-03***	0.5632 *+5.6e-04* *-5.6e-04*	1.1995 *+1.3e-03* *-1.3e-03*
			Filter(Extrap,0)	**3.394** ***+4.0e-02*** ***-3.5e-02***	0.14007 *+2.7e-03* *-2.5e-03*	0.6362 *+6.3e-04* *-6.1e-04*	1.2937 *+1.4e-03* *-1.5e-03*
			Filter(Extrap,1)	0.861 *+2.0e-02* *-2.6e-02*	0.85825 *+3.2e-03* *-2.7e-03*	0.5098 *+5.9e-04* *-6.1e-04*	1.1266 *+1.2e-03* *-1.4e-03*
			Filter(DLE,4)	1.509 *+3.2e-02* *-3.1e-02*	0.82407 *+3.3e-03* *-3.5e-03*	0.5436 *+6.1e-04* *-6.3e-04*	1.1581 *+1.3e-03* *-1.3e-03*
			Filter(DLE,8)	1.221 *+3.3e-02* *-2.8e-02*	0.85630 *+3.0e-03* *-3.6e-03*	0.5296 *+5.9e-04* *-6.2e-04*	1.1459 *+1.3e-03* *-1.3e-03*

continued on next page

Table B.3: Simulation Results for Background Wireless Traffic Scenario – continued

Bgnd. Packet Size (bytes)	Bgnd. Mean Send Period (s)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
4096	0.01	30	Queue(1, DropOldest)	3.102 *+3.2e-02* *-2.9e-02*	0.11263 *+2.6e-03* *-2.6e-03*	0.6075 *+5.3e-04* *-5.7e-04*	1.2747 *+1.5e-03* *-1.5e-03*
			Queue(10, DropOldest)	3.102 *+3.2e-02* *-2.9e-02*	0.11263 *+2.6e-03* *-2.6e-03*	0.6075 *+5.3e-04* *-5.7e-04*	1.2747 *+1.5e-03* *-1.5e-03*
			Queue(20, DropOldest)	3.102 *+3.2e-02* *-2.9e-02*	0.11263 *+2.6e-03* *-2.6e-03*	0.6075 *+5.3e-04* *-5.7e-04*	1.2747 *+1.5e-03* *-1.5e-03*
			Queue(50, DropOldest)	3.102 *+3.2e-02* *-2.9e-02*	0.11263 *+2.6e-03* *-2.6e-03*	0.6075 *+5.3e-04* *-5.7e-04*	1.2747 *+1.5e-03* *-1.5e-03*
			Filter(Extrap,0)	**4.547** ***+4.0e-02*** ***-4.6e-02***	**0.09415** ***+2.5e-03*** ***-2.2e-03***	0.6977 *+6.7e-04* *-7.2e-04*	1.3756 *+1.5e-03* *-1.5e-03*
			Filter(Extrap,1)	3.128 *+4.8e-02* *-4.9e-02*	0.67036 *+4.6e-03* *-4.7e-03*	0.6811 *+9.5e-04* *-9.8e-04*	1.3914 *+1.8e-03* *-2.0e-03*
			Filter(DLE,4)	2.484 *+3.9e-02* *-4.3e-02*	0.71184 *+4.0e-03* *-4.3e-03*	0.6131 *+7.5e-04* *-7.4e-04*	1.2844 *+1.6e-03* *-1.6e-03*
			Filter(DLE,8)	1.843 *+3.6e-02* *-3.5e-02*	0.77029 *+3.6e-03* *-3.7e-03*	0.5784 *+7.3e-04* *-7.7e-04*	1.2434 *+1.7e-03* *-1.7e-03*

continued on next page

Table B.3: Simulation Results for Background Wireless Traffic Scenario – continued

Bgnd. Packet Size (bytes)	Bgnd. Mean Send Period (s)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
4096	0.10	30	Queue(1, DropOldest)	2.002 *+2.5e-02* *-2.6e-02*	**0.15415** ***+2.6e-03*** ***-3.3e-03***	0.5588 *+5.3e-04* *-6.1e-04*	1.2049 *+1.3e-03* *-1.5e-03*
			Queue(10, DropOldest)	2.002 *+2.5e-02* *-2.6e-02*	**0.15415** ***+2.6e-03*** ***-3.3e-03***	0.5588 *+5.3e-04* *-6.1e-04*	1.2049 *+1.3e-03* *-1.5e-03*
			Queue(20, DropOldest)	2.002 *+2.5e-02* *-2.6e-02*	**0.15415** ***+2.6e-03*** ***-3.3e-03***	0.5588 *+5.3e-04* *-6.1e-04*	1.2049 *+1.3e-03* *-1.5e-03*
			Queue(50, DropOldest)	2.002 *+2.5e-02* *-2.6e-02*	**0.15415** ***+2.6e-03*** ***-3.3e-03***	0.5588 *+5.3e-04* *-6.1e-04*	1.2049 *+1.3e-03* *-1.5e-03*
			Filter(Extrap,0)	**3.067** ***+3.7e-02*** ***-3.7e-02***	**0.15296** ***+3.2e-03*** ***-2.9e-03***	0.6248 *+6.8e-04* *-6.6e-04*	1.2855 *+1.5e-03* *-1.4e-03*
			Filter(Extrap,1)	1.252 *+3.1e-02* *-2.9e-02*	0.84946 *+3.2e-03* *-3.2e-03*	0.5395 *+6.8e-04* *-7.0e-04*	1.1678 *+1.5e-03* *-1.4e-03*
			Filter(DLE,4)	1.327 *+3.5e-02* *-3.3e-02*	0.84568 *+3.6e-03* *-3.5e-03*	0.5345 *+6.6e-04* *-5.7e-04*	1.1488 *+1.3e-03* *-1.2e-03*
			Filter(DLE,8)	0.984 *+2.6e-02* *-2.8e-02*	0.88373 *+3.0e-03* *-2.7e-03*	0.5133 *+6.1e-04* *-5.5e-04*	1.1152 *+1.2e-03* *-1.2e-03*

B.4. Slow Update Frequency Scenario

Table B.4 shows the results from the Slow Update Frequency Scenario described in Section 7.5.4.

Table B.4: Simulation Results for Slow Update Frequency Scenario

Guidance Update Period (ticks)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
20	15	Queue(1, DropOldest)	4.997 *+5.1e-04* *-5.2e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.0877 *+5.5e-04* *-5.8e-04*	1.8203 *+9.2e-04* *-1.0e-03*
		Queue(10, DropOldest)	4.997 *+5.1e-04* *-5.2e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.0877 *+5.5e-04* *-5.8e-04*	1.8203 *+9.2e-04* *-1.0e-03*
		Queue(20, DropOldest)	4.997 *+5.1e-04* *-5.2e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.0877 *+5.5e-04* *-5.8e-04*	1.8203 *+9.2e-04* *-1.0e-03*
		Queue(50, DropOldest)	4.997 *+5.1e-04* *-5.2e-04*	**0.00000** ***+6.3e-05*** ***-0.0e+00***	1.0877 *+5.5e-04* *-5.8e-04*	1.8203 *+9.2e-04* *-1.0e-03*
		Filter(Extrap,0)	**5.000** ***+7.2e-04*** ***-8.6e-04***	**0.00000** ***+1.4e-04*** ***-0.0e+00***	1.0882 *+5.6e-04* *-6.0e-04*	1.8207 *+9.3e-04* *-9.7e-04*
		Filter(Extrap,1)	**5.000** ***+7.5e-04*** ***-8.4e-04***	**0.00000** ***+1.4e-04*** ***-0.0e+00***	1.0890 *+5.8e-04* *-5.6e-04*	1.8223 *+9.5e-04* *-9.4e-04*
		Filter(DLE,4)	**5.000** ***+7.4e-04*** ***-8.6e-04***	**0.00000** ***+1.4e-04*** ***-0.0e+00***	1.0891 *+5.6e-04* *-5.8e-04*	1.8225 *+9.4e-04* *-9.6e-04*
		Filter(DLE,8)	**5.000** ***+7.4e-04*** ***-8.3e-04***	**0.00000** ***+1.4e-04*** ***-0.0e+00***	1.0890 *+5.6e-04* *-5.9e-04*	1.8226 *+9.2e-04* *-1.0e-03*

continued on next page

Table B.4: Simulation Results for Slow Update Frequency Scenario – continued

Guidance Update Period (ticks)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
50	15	Queue(1, DropOldest)	4.995 *+2.1e-03* *-1.8e-03*	**0.00114** ***+7.7e-05*** ***-6.6e-05***	1.0601 *+4.1e-04* *-4.9e-04*	1.8126 *+9.0e-04* *-9.3e-04*
		Queue(10, DropOldest)	4.995 *+2.1e-03* *-1.8e-03*	**0.00114** ***+7.7e-05*** ***-6.6e-05***	1.0601 *+4.1e-04* *-4.9e-04*	1.8126 *+9.0e-04* *-9.3e-04*
		Queue(20, DropOldest)	4.995 *+2.1e-03* *-1.8e-03*	**0.00114** ***+7.7e-05*** ***-6.6e-05***	1.0601 *+4.1e-04* *-4.9e-04*	1.8126 *+9.0e-04* *-9.3e-04*
		Queue(50, DropOldest)	4.995 *+2.1e-03* *-1.8e-03*	**0.00114** ***+7.7e-05*** ***-6.6e-05***	1.0601 *+4.1e-04* *-4.9e-04*	1.8126 *+9.0e-04* *-9.3e-04*
		Filter(Extrap,0)	**5.008** ***+2.2e-03*** ***-2.2e-03***	**0.00139** ***+2.3e-04*** ***-2.0e-04***	1.0622 *+4.4e-04* *-5.1e-04*	1.8151 *+8.3e-04* *-9.0e-04*
		Filter(Extrap,1)	4.940 *+3.8e-03* *-4.0e-03*	0.01001 *+3.4e-04* *-3.4e-04*	1.1471 *+6.3e-04* *-6.1e-04*	1.9914 *+1.2e-03* *-1.1e-03*
		Filter(DLE,4)	4.945 *+4.3e-03* *-4.3e-03*	0.01185 *+3.8e-04* *-3.5e-04*	1.1415 *+5.9e-04* *-6.1e-04*	1.9910 *+1.1e-03* *-1.1e-03*
		Filter(DLE,8)	4.774 *+5.0e-03* *-5.4e-03*	0.04908 *+5.8e-04* *-5.5e-04*	1.1083 *+5.4e-04* *-5.0e-04*	1.9857 *+1.1e-03* *-1.1e-03*

continued on next page

Table B.4: Simulation Results for Slow Update Frequency Scenario – continued

Guidance Update Period (ticks)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
100	15	Queue(1, DropOldest)	**2.256** ***+6.3e-03*** ***-6.7e-03***	**0.05281** ***+5.8e-04*** ***-5.4e-04***	0.6775 *+2.4e-04* *-2.5e-04*	1.3105 *+5.1e-04* *-5.7e-04*
		Queue(10, DropOldest)	**2.256** ***+6.3e-03*** ***-6.7e-03***	**0.05281** ***+5.8e-04*** ***-5.4e-04***	0.6775 *+2.4e-04* *-2.5e-04*	1.3105 *+5.1e-04* *-5.7e-04*
		Queue(20, DropOldest)	**2.256** ***+6.3e-03*** ***-6.7e-03***	**0.05281** ***+5.8e-04*** ***-5.4e-04***	0.6775 *+2.4e-04* *-2.5e-04*	1.3105 *+5.1e-04* *-5.7e-04*
		Queue(50, DropOldest)	**2.256** ***+6.3e-03*** ***-6.7e-03***	**0.05281** ***+5.8e-04*** ***-5.4e-04***	0.6775 *+2.4e-04* *-2.5e-04*	1.3105 *+5.1e-04* *-5.7e-04*
		Filter(Extrap,0)	1.568 *+5.3e-03* *-6.2e-03*	0.18317 *+1.2e-03* *-1.1e-03*	0.6033 *+2.0e-04* *-2.0e-04*	1.2167 *+4.6e-04* *-4.6e-04*
		Filter(Extrap,1)	0.514 *+4.4e-03* *-4.4e-03*	0.87460 *+1.0e-03* *-1.1e-03*	0.4984 *+1.5e-04* *-1.6e-04*	1.0717 *+2.9e-04* *-3.1e-04*
		Filter(DLE,4)	0.456 *+4.2e-03* *-3.9e-03*	0.88988 *+9.0e-04* *-9.4e-04*	0.4918 *+1.4e-04* *-1.4e-04*	1.0627 *+2.7e-04* *-2.9e-04*
		Filter(DLE,8)	0.401 *+3.9e-03* *-4.1e-03*	0.90159 *+9.0e-04* *-8.6e-04*	0.4862 *+1.3e-04* *-1.3e-04*	1.0554 *+2.7e-04* *-2.7e-04*

continued on next page

Table B.4: Simulation Results for Slow Update Frequency Scenario – continued

Guidance Update Period (ticks)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
200	15	Queue(1, DropOldest)	**4.804** ***+4.4e-03*** ***-4.0e-03***	**0.03084** ***+4.4e-04*** ***-4.7e-04***	1.0598 *+4.7e-04* *-4.3e-04*	1.8695 *+9.1e-04* *-9.0e-04*
		Queue(10, DropOldest)	**4.804** ***+4.4e-03*** ***-4.0e-03***	**0.03084** ***+4.4e-04*** ***-4.7e-04***	1.0598 *+4.7e-04* *-4.3e-04*	1.8695 *+9.1e-04* *-9.0e-04*
		Queue(20, DropOldest)	**4.804** ***+4.4e-03*** ***-4.0e-03***	**0.03084** ***+4.4e-04*** ***-4.7e-04***	1.0598 *+4.7e-04* *-4.3e-04*	1.8695 *+9.1e-04* *-9.0e-04*
		Queue(50, DropOldest)	**4.804** ***+4.4e-03*** ***-4.0e-03***	**0.03084** ***+4.4e-04*** ***-4.7e-04***	1.0598 *+4.7e-04* *-4.3e-04*	1.8695 *+9.1e-04* *-9.0e-04*
		Filter(Extrap,0)	4.649 *+4.7e-03* *-5.0e-03*	0.04374 *+6.0e-04* *-6.3e-04*	1.0368 *+4.3e-04* *-4.3e-04*	1.8351 *+9.9e-04* *-9.1e-04*
		Filter(Extrap,1)	3.354 *+6.8e-03* *-7.1e-03*	0.23670 *+1.2e-03* *-1.2e-03*	0.8828 *+4.4e-04* *-4.7e-04*	1.6551 *+1.0e-03* *-1.1e-03*
		Filter(DLE,4)	3.335 *+6.4e-03* *-7.5e-03*	0.24604 *+1.2e-03* *-1.2e-03*	0.8800 *+4.4e-04* *-4.7e-04*	1.6469 *+9.8e-04* *-9.7e-04*
		Filter(DLE,8)	2.995 *+7.6e-03* *-7.2e-03*	0.31358 *+1.3e-03* *-1.3e-03*	0.8416 *+4.5e-04* *-5.1e-04*	1.6017 *+8.8e-04* *-9.4e-04*

continued on next page

Table B.4: Simulation Results for Slow Update Frequency Scenario – continued

Guidance Update Period (ticks)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
500	15	Queue(1, DropOldest)	**4.084** ***+6.3e-03*** ***-6.5e-03***	**0.06999** ***+7.7e-04*** ***-7.5e-04***	0.9608 *+3.9e-04* *-4.3e-04*	1.7596 *+9.0e-04* *-9.8e-04*
		Queue(10, DropOldest)	**4.084** ***+6.3e-03*** ***-6.5e-03***	**0.06999** ***+7.7e-04*** ***-7.5e-04***	0.9608 *+3.9e-04* *-4.3e-04*	1.7596 *+9.0e-04* *-9.8e-04*
		Queue(20, DropOldest)	**4.084** ***+6.3e-03*** ***-6.5e-03***	**0.06999** ***+7.7e-04*** ***-7.5e-04***	0.9608 *+3.9e-04* *-4.3e-04*	1.7596 *+9.0e-04* *-9.8e-04*
		Queue(50, DropOldest)	**4.084** ***+6.3e-03*** ***-6.5e-03***	**0.06999** ***+7.7e-04*** ***-7.5e-04***	0.9608 *+3.9e-04* *-4.3e-04*	1.7596 *+9.0e-04* *-9.8e-04*
		Filter(Extrap,0)	3.875 *+6.3e-03* *-6.0e-03*	0.09605 *+7.9e-04* *-8.9e-04*	0.9380 *+4.0e-04* *-4.3e-04*	1.7390 *+9.2e-04* *-1.0e-03*
		Filter(Extrap,1)	3.875 *+6.3e-03* *-6.0e-03*	0.09605 *+7.9e-04* *-8.9e-04*	0.9380 *+4.0e-04* *-4.3e-04*	1.7390 *+9.2e-04* *-1.0e-03*
		Filter(DLE,4)	3.875 *+6.3e-03* *-6.0e-03*	0.09605 *+7.9e-04* *-8.9e-04*	0.9380 *+4.0e-04* *-4.3e-04*	1.7390 *+9.2e-04* *-1.0e-03*
		Filter(DLE,8)	3.875 *+6.3e-03* *-6.0e-03*	0.09605 *+7.9e-04* *-8.9e-04*	0.9380 *+4.0e-04* *-4.3e-04*	1.7390 *+9.2e-04* *-1.0e-03*

continued on next page

Table B.4: Simulation Results for Slow Update Frequency Scenario – continued

Guidance Update Period (ticks)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
20	30	Queue(1, DropOldest)	9.985 *+1.4e-03* *-1.3e-03*	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.0875 *+4.4e-04* *-4.3e-04*	1.8212 *+7.3e-04* *-7.2e-04*
		Queue(10, DropOldest)	9.985 *+1.4e-03* *-1.3e-03*	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.0875 *+4.4e-04* *-4.3e-04*	1.8212 *+7.3e-04* *-7.2e-04*
		Queue(20, DropOldest)	9.985 *+1.4e-03* *-1.3e-03*	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.0875 *+4.4e-04* *-4.3e-04*	1.8212 *+7.3e-04* *-7.2e-04*
		Queue(50, DropOldest)	9.985 *+1.4e-03* *-1.3e-03*	**0.00000** ***+7.0e-05*** ***-0.0e+00***	1.0875 *+4.4e-04* *-4.3e-04*	1.8212 *+7.3e-04* *-7.2e-04*
		Filter(Extrap,0)	**9.990** ***+1.6e-03*** ***-1.7e-03***	**0.00000** ***+1.5e-04*** ***-0.0e+00***	1.0857 *+4.2e-04* *-3.5e-04*	1.8176 *+7.1e-04* *-6.0e-04*
		Filter(Extrap,1)	**9.990** ***+1.7e-03*** ***-1.8e-03***	**0.00000** ***+1.5e-04*** ***-0.0e+00***	1.1044 *+3.9e-04* *-3.8e-04*	1.8572 *+6.3e-04* *-6.4e-04*
		Filter(DLE,4)	**9.990** ***+1.7e-03*** ***-1.8e-03***	**0.00000** ***+1.5e-04*** ***-0.0e+00***	1.1064 *+4.3e-04* *-3.8e-04*	1.8617 *+7.6e-04* *-6.7e-04*
		Filter(DLE,8)	**9.990** ***+1.8e-03*** ***-1.8e-03***	**0.00000** ***+1.5e-04*** ***-2.8e-07***	1.1082 *+4.1e-04* *-3.6e-04*	1.8669 *+7.1e-04* *-6.3e-04*

continued on next page

Table B.4: Simulation Results for Slow Update Frequency Scenario – continued

Guidance Update Period (ticks)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
50	30	Queue(1, DropOldest)	9.983 *+2.6e-03* *-2.5e-03*	**0.00012** ***+7.2e-05*** ***-6.2e-05***	1.0726 *+3.5e-04* *-3.6e-04*	1.8063 *+6.3e-04* *-6.0e-04*
		Queue(10, DropOldest)	9.983 *+2.6e-03* *-2.5e-03*	**0.00012** ***+7.2e-05*** ***-6.2e-05***	1.0726 *+3.5e-04* *-3.6e-04*	1.8063 *+6.3e-04* *-6.0e-04*
		Queue(20, DropOldest)	9.983 *+2.6e-03* *-2.5e-03*	**0.00012** ***+7.2e-05*** ***-6.2e-05***	1.0726 *+3.5e-04* *-3.6e-04*	1.8063 *+6.3e-04* *-6.0e-04*
		Queue(50, DropOldest)	9.983 *+2.6e-03* *-2.5e-03*	**0.00012** ***+7.2e-05*** ***-6.2e-05***	1.0726 *+3.5e-04* *-3.6e-04*	1.8063 *+6.3e-04* *-6.0e-04*
		Filter(Extrap,0)	**10.015** ***+2.8e-03*** ***-3.0e-03***	**0.00011** ***+2.2e-04*** ***-1.1e-04***	1.0716 *+3.4e-04* *-3.4e-04*	1.8007 *+6.1e-04* *-6.4e-04*
		Filter(Extrap,1)	**10.010** ***+5.2e-03*** ***-5.5e-03***	0.00066 *+2.3e-04* *-1.9e-04*	1.2558 *+5.8e-04* *-5.9e-04*	2.1771 *+1.0e-03* *-9.8e-04*
		Filter(DLE,4)	**10.008** ***+7.3e-03*** ***-7.9e-03***	0.00095 *+2.3e-04* *-1.9e-04*	1.2518 *+5.6e-04* *-4.8e-04*	2.2418 *+9.8e-04* *-9.7e-04*
		Filter(DLE,8)	9.987 *+1.1e-02* *-1.0e-02*	0.00998 *+3.5e-04* *-3.3e-04*	1.1986 *+4.3e-04* *-4.5e-04*	2.1834 *+1.0e-03* *-9.6e-04*

continued on next page

Table B.4: Simulation Results for Slow Update Frequency Scenario – continued

Guidance Update Period (ticks)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
100	30	Queue(1, DropOldest)	**2.231** ***+7.5e-03*** ***-7.0e-03***	**0.11652** ***+7.3e-04*** ***-7.6e-04***	0.5548 *+1.4e-04* *-1.4e-04*	1.1775 *+3.6e-04* *-3.8e-04*
		Queue(10, DropOldest)	**2.231** ***+7.5e-03*** ***-7.0e-03***	**0.11652** ***+7.3e-04*** ***-7.6e-04***	0.5548 *+1.4e-04* *-1.4e-04*	1.1775 *+3.6e-04* *-3.8e-04*
		Queue(20, DropOldest)	**2.231** ***+7.5e-03*** ***-7.0e-03***	**0.11652** ***+7.3e-04*** ***-7.6e-04***	0.5548 *+1.4e-04* *-1.4e-04*	1.1775 *+3.6e-04* *-3.8e-04*
		Queue(50, DropOldest)	**2.231** ***+7.5e-03*** ***-7.0e-03***	**0.11652** ***+7.3e-04*** ***-7.6e-04***	0.5548 *+1.4e-04* *-1.4e-04*	1.1775 *+3.6e-04* *-3.8e-04*
		Filter(Extrap,0)	1.907 *+6.3e-03* *-6.3e-03*	0.17172 *+9.8e-04* *-1.0e-03*	0.5380 *+1.2e-04* *-1.2e-04*	1.1523 *+3.5e-04* *-3.5e-04*
		Filter(Extrap,1)	0.288 *+4.3e-03* *-4.2e-03*	0.95501 *+5.4e-04* *-5.8e-04*	0.4594 *+6.6e-05* *-6.3e-05*	1.0237 *+1.7e-04* *-1.4e-04*
		Filter(DLE,4)	0.270 *+4.6e-03* *-4.3e-03*	0.95005 *+5.4e-04* *-6.2e-04*	0.4586 *+7.1e-05* *-6.3e-05*	1.0225 *+1.6e-04* *-1.6e-04*
		Filter(DLE,8)	0.253 *+4.4e-03* *-4.1e-03*	0.96464 *+5.1e-04* *-5.5e-04*	0.4581 *+6.7e-05* *-6.2e-05*	1.0217 *+1.5e-04* *-1.5e-04*

continued on next page

Table B.4: Simulation Results for Slow Update Frequency Scenario – continued

Guidance Update Period (ticks)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
200	30	Queue(1, DropOldest)	**4.337** ***+1.3e-02*** ***-1.3e-02***	**0.23404** ***+1.1e-03*** ***-1.3e-03***	0.7380 *+2.8e-04* *-2.9e-04*	1.4627 *+5.6e-04* *-6.2e-04*
		Queue(10, DropOldest)	**4.337** ***+1.3e-02*** ***-1.3e-02***	**0.23404** ***+1.1e-03*** ***-1.3e-03***	0.7380 *+2.8e-04* *-2.9e-04*	1.4627 *+5.6e-04* *-6.2e-04*
		Queue(20, DropOldest)	**4.337** ***+1.3e-02*** ***-1.3e-02***	**0.23404** ***+1.1e-03*** ***-1.3e-03***	0.7380 *+2.8e-04* *-2.9e-04*	1.4627 *+5.6e-04* *-6.2e-04*
		Queue(50, DropOldest)	**4.337** ***+1.3e-02*** ***-1.3e-02***	**0.23404** ***+1.1e-03*** ***-1.3e-03***	0.7380 *+2.8e-04* *-2.9e-04*	1.4627 *+5.6e-04* *-6.2e-04*
		Filter(Extrap,0)	2.776 *+1.0e-02* *-9.6e-03*	0.29667 *+1.4e-03* *-1.3e-03*	0.6408 *+2.5e-04* *-2.8e-04*	1.3376 *+5.3e-04* *-5.4e-04*
		Filter(Extrap,1)	2.449 *+8.1e-03* *-8.1e-03*	0.56917 *+1.2e-03* *-1.1e-03*	0.6310 *+2.7e-04* *-2.7e-04*	1.3567 *+5.5e-04* *-5.6e-04*
		Filter(DLE,4)	2.348 *+7.6e-03* *-8.0e-03*	0.57945 *+1.1e-03* *-1.1e-03*	0.6248 *+2.8e-04* *-2.6e-04*	1.3483 *+6.5e-04* *-6.4e-04*
		Filter(DLE,8)	2.386 *+8.9e-03* *-8.4e-03*	0.58058 *+1.1e-03* *-1.2e-03*	0.6278 *+2.7e-04* *-2.8e-04*	1.3535 *+6.7e-04* *-6.6e-04*

continued on next page

Table B.4: Simulation Results for Slow Update Frequency Scenario – continued

Guidance Update Period (ticks)	Car Count	Mechanism	Flow (cells/tick)	Motionless Ratio (unitless)	EFC (mL/s)	NFR (unitless)
500	30	Queue(1, DropOldest)	**7.433** ***+1.3e-02*** ***-1.4e-02***	**0.08550** ***+7.1e-04*** ***-7.8e-04***	0.9184 *+3.0e-04* *-3.3e-04*	1.7227 *+6.2e-04* *-7.4e-04*
		Queue(10, DropOldest)	**7.433** ***+1.3e-02*** ***-1.4e-02***	**0.08550** ***+7.1e-04*** ***-7.8e-04***	0.9184 *+3.0e-04* *-3.3e-04*	1.7227 *+6.2e-04* *-7.4e-04*
		Queue(20, DropOldest)	**7.433** ***+1.3e-02*** ***-1.4e-02***	**0.08550** ***+7.1e-04*** ***-7.8e-04***	0.9184 *+3.0e-04* *-3.3e-04*	1.7227 *+6.2e-04* *-7.4e-04*
		Queue(50, DropOldest)	**7.433** ***+1.3e-02*** ***-1.4e-02***	**0.08550** ***+7.1e-04*** ***-7.8e-04***	0.9184 *+3.0e-04* *-3.3e-04*	1.7227 *+6.2e-04* *-7.4e-04*
		Filter(Extrap,0)	6.363 *+1.4e-02* *-1.3e-02*	0.13172 *+1.0e-03* *-1.0e-03*	0.8571 *+3.3e-04* *-3.0e-04*	1.6707 *+7.6e-04* *-7.4e-04*
		Filter(Extrap,1)	6.363 *+1.4e-02* *-1.3e-02*	0.13172 *+1.0e-03* *-1.0e-03*	0.8571 *+3.3e-04* *-3.0e-04*	1.6707 *+7.6e-04* *-7.4e-04*
		Filter(DLE,4)	6.363 *+1.4e-02* *-1.3e-02*	0.13172 *+1.0e-03* *-1.0e-03*	0.8571 *+3.3e-04* *-3.0e-04*	1.6707 *+7.6e-04* *-7.4e-04*
		Filter(DLE,8)	6.363 *+1.4e-02* *-1.3e-02*	0.13172 *+1.0e-03* *-1.0e-03*	0.8571 *+3.3e-04* *-3.0e-04*	1.6707 *+7.6e-04* *-7.4e-04*

Appendix C

Traffic Case Study Inter-arrival Times

This appendix summarizes the inter-arrival times of the guidance packets observed at the gateway in the vehicle. The results are grouped by the various scenarios tested in the traffic case study, as described in Section 7.5. The inter-arrival times have been aggregated over all experiments into a box plot (see Figure 6.1 for a description of the boxplot statistics). In some cases, multiple plots of the same data are shown with the axes truncated differently to show the results at different time scales.

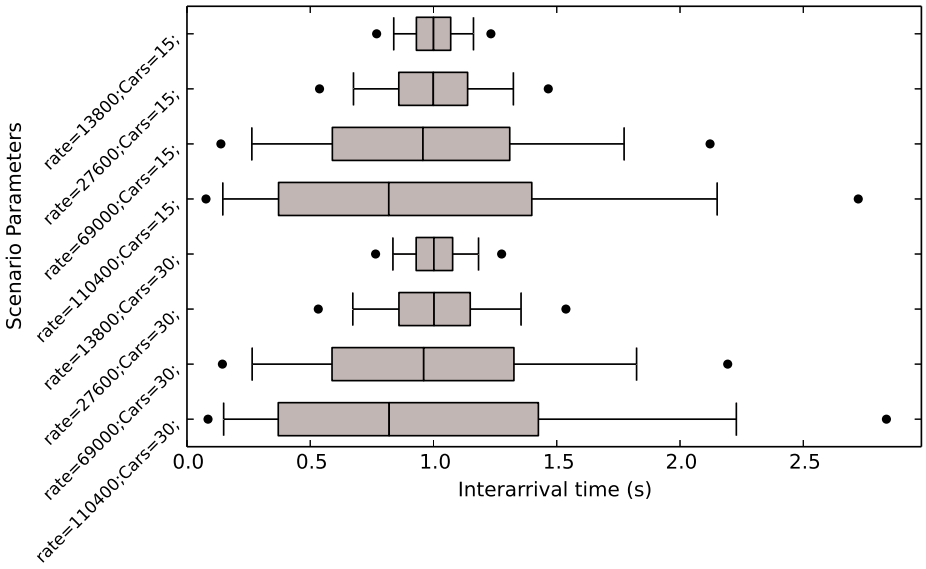


Figure C.1. Aggregate Inter-arrival Times for Internet Link Congestion Scenario

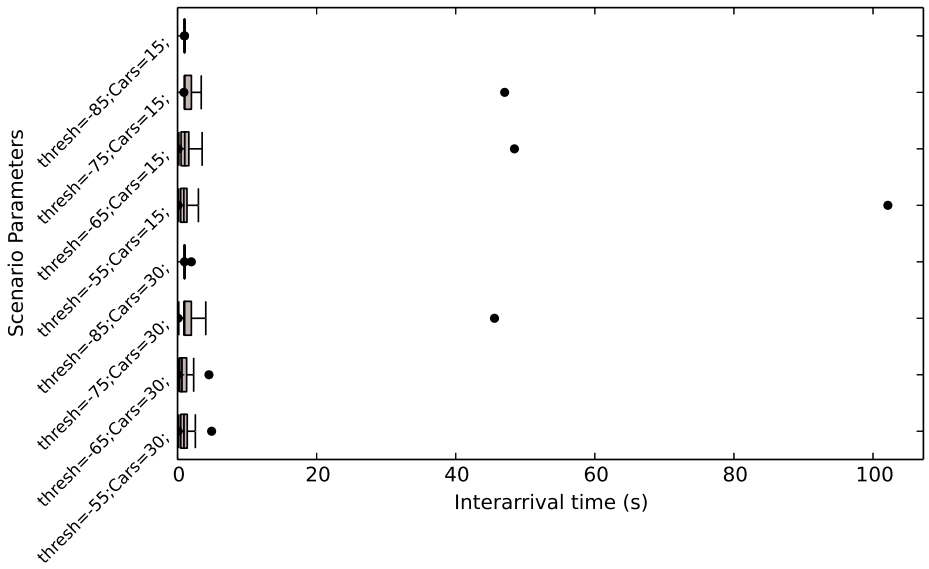


Figure C.2. Aggregate Inter-arrival Times for Noisy Wireless Traffic Congestion Scenario

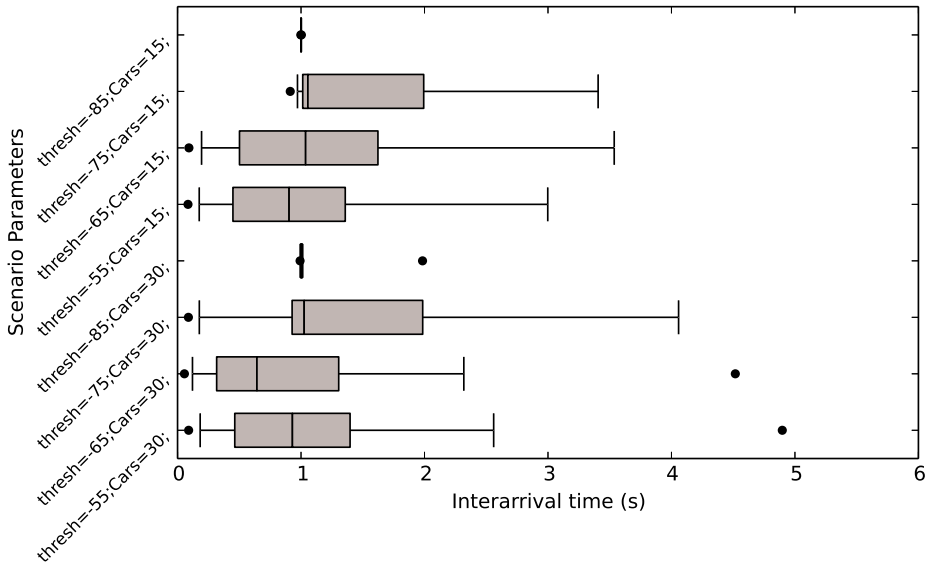


Figure C.3. Aggregate Inter-arrival Times for Noisy Wireless Traffic Congestion Scenario — Plot is truncated to 6 seconds on the X axis to show details at that timescale.

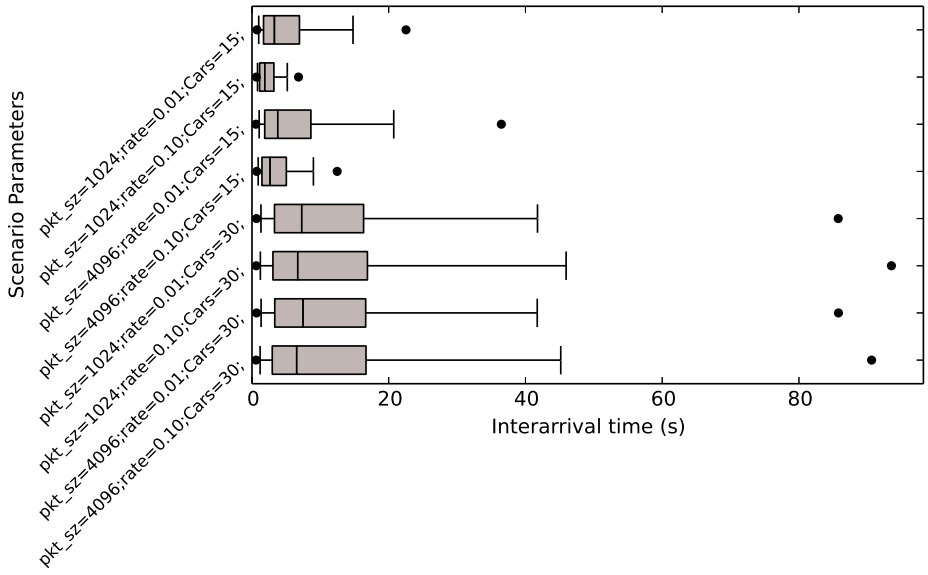


Figure C.4. Aggregate Inter-arrival Times for Wireless Traffic Congestion Scenario

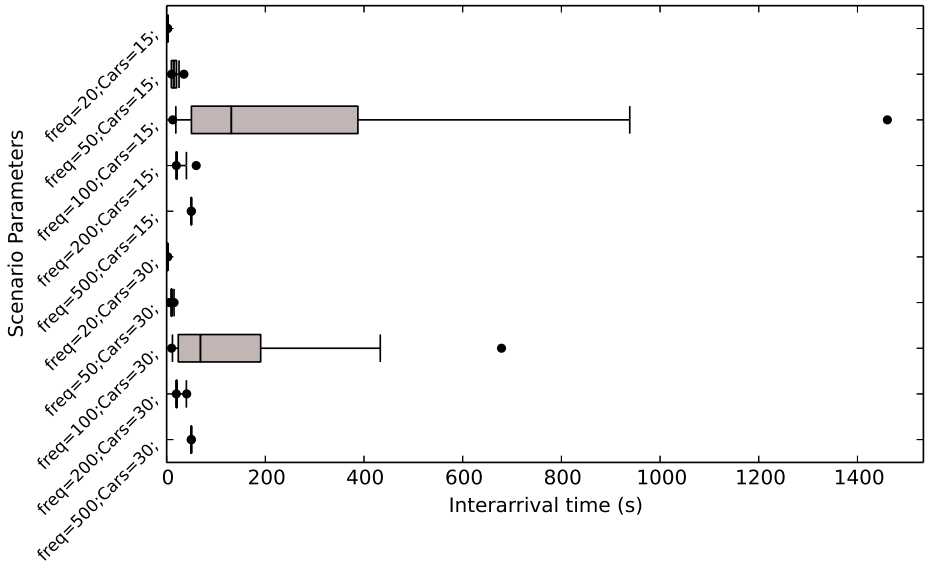


Figure C.5. Aggregate Inter-arrival Times for Slow Update Frequency Scenario

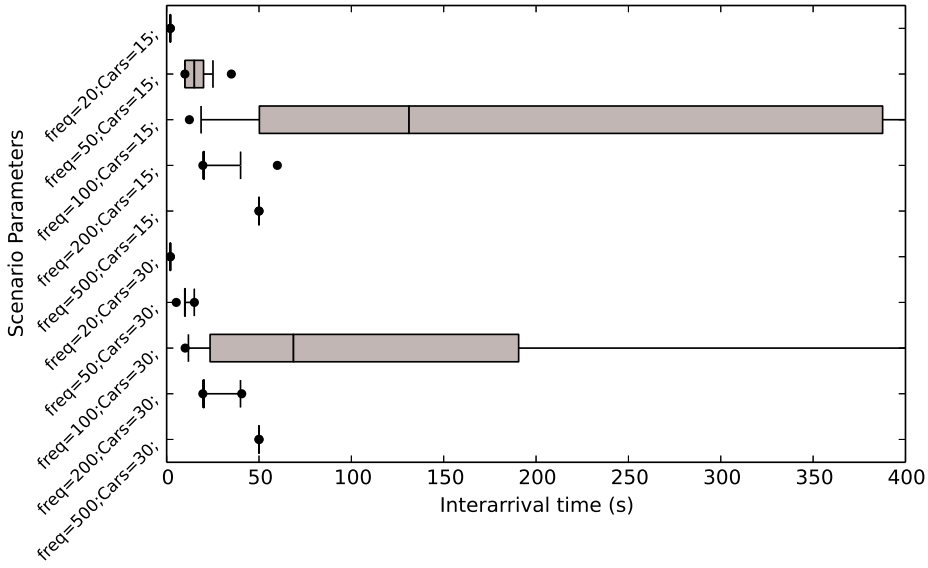


Figure C.6. Aggregate Inter-arrival Times for Slow Update Frequency Scenario — Plot is truncated to 400 seconds on the X axis to show details at that timescale.

Bibliography

[1] OnStar, LLC., "Onstar 801a," Nov. 2008. [Online]. Available: https://www.onstarfulfillment.com/pdf/EN_OnStar_Gen8_Manual.pdf [Accessed: 29 Nov. 2013].

[2] BMW Group, "BMW ConnectedDrive," 2009. [Online]. Available: http://www.bmw.com/com/en/owners/navigation/incarinternet.html [Accessed: 29 Nov. 2013].

[3] Ford Motor Company, "Popular Ford SYNC system expanded; new Sirius Travel Link takes navigation to new level," 2009. [Online]. Available: http://support.ford.com/sync-technology/sync-services-overview-sync-myford-touch [Accessed: 29 Nov. 2013].

[4] Y. Zhao, "Telematics: safe and fun driving," *Intelligent Systems, IEEE*, vol. 17, no. 1, pp. 10–14, Jan./Feb. 2002.

[5] M. Becchi, M. Franklin, and P. Crowley, "A workload for evaluating deep packet inspection architectures," in *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*, Sept. 2008, pp. 79 –89.

[6] D. Wu, Y. Hou, W. Zhu, Y.-Q. Zhang, and J. Peha, "Streaming video over the internet: approaches and directions," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 11, no. 3, pp. 282 –300, Mar. 2001.

[7] Riverbed Technology, "OPNET® modeler version 16.0A PL7 [software]," 1986–2013. [Online]. Available: http://www.riverbed.com/products-solutions/products/network-performance-management/network-planning-simulation/Network-Simulation.html [Accessed: 29 Nov. 2013].

[8] M. Chao-Qun, H. Hai-Jun, and T. Tie-Qiao, "Improving urban traffic by velocity guidance," in *Intelligent Computation Technology and Automation (ICICTA), 2008 International Conference on*, vol. 2, Oct. 2008, pp. 383 –387.

[9] T. Faber, "ACC: using active networking to enhance feedback congestion control mechanisms," *Network, IEEE*, vol. 12, no. 3, pp. 61–65, 1998.

[10] J. Ray and P. Koopman, "Data management mechanisms for embedded system gateways," in *Dependable Systems and Networks, 2009. DSN '09. IEEE/IFIP International Conference on*, July 2009, pp. 175–184.

[11] C. A. Sunshine, "Interconnection of computer networks," in *Computer Networks*, vol. 1, 1977.

[12] W. C. Feng, K. Shin, D. Kandlur, and D. Saha, "The BLUE active queue management algorithms," *Networking, IEEE/ACM Transactions on*, vol. 10, no. 4, pp. 513–528, Aug. 2002.

[13] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *Networking, IEEE/ACM Transactions on*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

[14] Q. Huang, J. S. Smith, and T. Li, "Web-based distributed embedded gateway system design," in *WI '06: Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 905–908.

[15] J. Kim, S. Seo, T. Moon, K. Kwon, J. Jeon, and S. Hwang, "A method for improving the reliability of the gateway system by using OSEK and duplication scheme," in *Industrial Technology, 2008. ICIT 2008. IEEE International Conference on*, April 2008, pp. 1–6.

[16] L. Hui, Z. Hao, P. Daogang, and H. Wen, "Design and application of communication gateway based on FlexRay and CAN," in *Electronic Computer Technology, 2009 International Conference on*, Feb. 2009, pp. 664–668.

[17] R. Manoharan, S. Rajarajan, S. Sashtinathan, and K. Sriram, "A novel multi-hop B3G architecture for adaptive gateway management in heterogeneous wireless networks," in *Wireless and Mobile Computing, Networking and Communications, 2009. WIMOB 2009. IEEE International Conference on*, Oct. 2009, pp. 447–452.

[18] R. Levin, E. Cohen, W. Corwin, F. Pollack, and W. Wulf, "Policy/mechanism separation in Hydra," *SIGOPS Oper. Syst. Rev.*, vol. 9, no. 5, pp. 132–140, 1975.

[19] P. Samarati and S. de Vimercati, "Access control: Policies, models, and mechanisms," in *Foundations of Security Analysis and Design*, ser. Lecture Notes in Computer Science, R. Focardi and R. Gorrieri, Eds. Springer Berlin / Heidelberg, 2001, vol. 2171, pp. 137–196.

[20] T. Lakshman and U. Madhow, "The performance of TCP/IP for networks with high bandwidth-delay products and random loss," *Networking, IEEE/ACM Transactions on*, vol. 5, no. 3, pp. 336 –350, June 1997.

[21] IEEE Std 802.11n, "Part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications," 2009, iSBN 0-7381-3397-1.

[22] H. Holma and A. Toskala, *HSDPA/HSUPA for UMTS: High Speed Radio Access for Mobile Communications*. John Wiley & Sons, 2006.

[23] V. Frost and B. Melamed, "Traffic modeling for telecommunications networks," *Communications Magazine, IEEE*, vol. 32, no. 3, pp. 70 –81, Mar. 1994.

[24] FlexRay-Consortium, "FlexRay communications system, protocol specification, version 2.0." [Online]. Available: http://www.flexray.com/specification_request.php [Accessed: 17 Sept. 2009].

[25] TTA-Group, "Time-triggered protocol TTP/C, high-level specification document, protocol version 1.1," 2003. [Online]. Available: http://www.ttagroup.org/technology/specification.htm [Accessed: 17 Sept. 2009].

[26] Robert Bosch, GmbH, "CAN specification, version 2.0," Postfach 50, D-7000 Stuttgart 1, 1991.

[27] B. Müller, T. Führer, F. Hartwich, R. Hugel, H. Weiler, and Robert Bosch GmbH, "Fault tolerant TTCAN networks," in *Proceedings 8th International CAN Conference*, 2002.

[28] H. Kopetz, "Event-triggered versus time-triggered real-time systems," in *Operating Systems of the 90s and Beyond*, ser. Lecture Notes in Computer Science, A. Karshmer and J. Nehmer, Eds. Springer Berlin / Heidelberg, 1991, vol. 563, pp. 86–101, 10.1007/BFb0024530.

[29] J. Kingman, "The first Erlang century–and the next," *Queueing Systems*, vol. 63, pp. 3–12, 2009, 10.1007/s11134-009-9147-4.

[30] J. Lehoczky, "Scheduling communication networks carrying real-time traffic," in *Real-Time Systems Symposium, 1998. Proceedings., The 19th IEEE*, Dec. 1998, pp. 470 –479.

[31] H. Sirisena, A. Haider, and K. Pawlikowski, "Auto-tuning RED for accurate queue control," *Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE*, vol. 2, pp. 2010–2015 vol.2, Nov. 2002.

[32] N. Boustany, M. Folkerts, K. Rao, A. Ray, L. Troxel, and Z. Zhang, "A simulation based methodology for analyzing network-based intelligent vehicle control systems," in *Intelligent Vehicles '92 Symposium., Proceedings of the*, June 1992, pp. 138 –143.

[33] W. Leutzbach, *Introduction to the theory of traffic flow*. Springer, 1972.

[34] K. Nagel and M. Schreckenberg, "A cellular automaton model for freeway traffic," *Journal de Physique I*, vol. 2, pp. 2221–2229, Dec. 1992.

[35] Oracle, "Java platform^{TM}, standard edition 6 API specification." [Online]. Available: http://docs.oracle.com/javase/6/docs/api/ [Accessed: 25 Nov. 2013].

[36] J. Samuel, "Emission related diagnostic services," *IEEE Communication Standards for European On-Board-Diagnostics Seminar (Ref. No. 1998/294)*, pp. 10/1–10/6, Feb. 1998.

[37] H. Jiang and C. Dovrolis, "Why is the internet traffic bursty in short time scales?" in *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. New York, NY, USA: ACM Press, 2005, pp. 241–252.

[38] J. Nilsson, "Real-time control systems with delays," Ph.D. dissertation, Lund Institute of Technology, 1998.

[39] T. Lakshman, A. Neidhardt, and T. Ott, "The drop from front strategy in TCP and in TCP over ATM," *INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE*, vol. 3, pp. 1242–1250 vol.3, Mar. 1996.

[40] A. Mankin, "Random drop congestion control," *SIGCOMM Comput. Commun. Rev.*, vol. 20, no. 4, pp. 1–7, 1990.

[41] B. Torkian, "Neville's algorithm." [Online]. Available: http://www.torkian.info/Site/Research/Entries/2008/2/29_Nevilles_algorithm_Java_Code_files/N.java [Accessed: 17 September 2009].

[42] P. Ettler, M. Kárný, and P. Nedoma, "Model mixing for long-term extrapolation," in *Modelling and Simulation, Proceedings of the 6th EUROSIM Congress on*, 2007.

[43] F. Gustafsson, *Adaptive Filtering and Change Detection*. Wiley, 2000.

[44] R. Franke, "Locally determined smooth interpolation at irregularly spaced points in several variables," *IMA Journal of Applied Mathematics*, vol. 19, no. 4, pp. 471–482, 1977.

[45] MATLAB, *version R2013a*. Natick, Massachusetts: The MathWorks Inc., 2013.

[46] E. Brockfeld, R. Barlovic, A. Schadschneider, and M. Schreckenberg, "Optimizing traffic lights in a cellular automaton model for city traffic," *Phys. Rev. E*, vol. 64, p. 056132, Oct. 2001.

[47] D. Biggs and R. Akcelik, "An energy-related model of instantaneous fuel consumption," *Traffic engineering & control*, vol. 27, no. 6, pp. 320–325, 1986.

[48] G. Song, L. Yu, and Z. Wang, "Aggregate fuel consumption model of light-duty vehicles for evaluating effectiveness of traffic management strategies on fuels," *Journal of Transportation Engineering*, vol. 135, no. 9, pp. 611–618, 2009.

[49] J. Jiménez-Palacios, "Understanding and quantifying motor vehicle emissions with vehicle specific power and TILDAS remote sensing." Ph.D. dissertation, Massachusetts Institute of Technology, 1999.

[50] S. Floyd and V. Paxson, "Difficulties in simulating the internet," *Networking, IEEE/ACM Transactions on*, vol. 9, no. 4, pp. 392 –403, Aug. 2001.

[51] J.-C. Bolot, "End-to-end packet delay and loss behavior in the internet," in *Conference proceedings on Communications architectures, protocols and applications*, ser. SIGCOMM '93. New York, NY, USA: ACM, 1993, pp. 289–298.

[52] L. Zhang, S. Shenker, and D. D. Clark, "Observations on the dynamics of a congestion control algorithm: the effects of two-way traffic," in *Proceedings of the conference on Communications architecture & protocols*, ser. SIGCOMM '91. New York, NY, USA: ACM, 1991, pp. 133–147.

[53] B. Efron, *5. The Bootstrap*. Philadephia, PA: Society for Industrial and Applied Mathematics, 1982, ch. 5, pp. 27–36.

[54] Z. Segall, D. Vrsalovic, D. Siewiorek, D. Yaskin, J. Kownacki, J. Barton, R. Dancey, A. Robinson, and T. Lin, "Fiat-fault injection based automated testing environment," in *Fault-Tolerant Computing, 1988. FTCS-18, Digest of Papers., Eighteenth International Symposium on*, 1988, pp. 102–107.