ГАРАНТИРОВАННОЕ ВРЕМЯ ДОСТАВКИ СРОЧНЫХ СООБЩЕНИЙ В РАСПРЕДЕЛЕННЫХ СЕТЯХ ПОВЫШЕННОЙ НАДЕЖНОСТИ В РЕАЛЬНОМ ВРЕМЕНИ.

Ken Tindell, Alan Burns
Real-Time Systems Research Group,
Department of Computer Science,
University of York, YO1 5DD, England

(С авторами можно связаться через *e-mail: ken@minster.york.ac.uk*; копии технических докладов университета Йорка доступны через FTP из *minster.york.ac.uk* в директории pub/realtime/papers)

Предисловие.

Представленный анализ позволяет предсказывать неблагоприятные времена доставки сообщений в сети контроллера (controller area network - CAN). Анализ иллюстрируется примерами контроллера Intel 82527, приведены результаты стандартных тестов SAE. В этих тестах используется около 53 типов сообщений; которые анализируются для различных скоростей передачи. Представлена техника, позволяющая улучшить временные показатели системы CAN. В частности оценено влияние сообщения "piggybacking". Статья завершается рассмотрением восстановления при сбоях, и представляет каркас в который могут включаться и анализироваться другие модели сбоев (с точки зрения их влияния на время доставки сообщения).

1. Введение

Последнее направление во многих управляющих системах -- соединить распределенные

элементы управляющей системы через коллективную шину широковещательной рассылки вместо использования двухточечных (point-to-point) связей [1]. Между общей шиной и двухточечными связями существует несколько фундаментальных различий. Во-первых, поскольку шина разделяется между несколькими крупными элементами системы, имеет место конкуренция за доступ к шине, которая должна проводиться по некоторому протоколу. Во-вторых, передача сигнала или данных не происходит мгновенно; различные сигналы могут иметь различные требования ко времени доставки. Следовательно есть потребность в алгоритмах для удовлетворения всех требований к временам доставки.

Существует несколько шинных технологий, но в этой статье мы рассматриваем сеть контроллера (CAN) [2], и для сравнения TTP(time triggered protocol) [3]. Эти две шины отличаются следующим: в CAN используется динамический метод, использующий приоритетный алгоритм для определения которой из подключенных станций разрешается посылать данные в шину. TTP использует статический метод, где каждой станции предоставляется фиксированный квант времени, когда она может передавать данные. Хотя CAN очень хороша при передаче наиболее неотложных данных, она не в состоянии гарантировать выполнение условаий для менее неотложных данных [3, 5]. Это не случайно: динамический алгоритм планирования, использованный CAN фактически идентичен алгоритму планирования обычно используемому в системах реального времени для планирования вычислений в процессорах. Анализ распределения времени в таких системах может быть применен почти без изменения к проблеме определения наихудшего времени доставки данного сообщения поставленного в очередь для передачи в CAN.

Эта статья воспроизводит существующий анализ распределения времени процессора, и показывает как этот анализ может быть применен к CAN. Для того, чтобы сохранить анализ точным, должны быть известными детали реализации контроллеров CAN, и поэтому в этой статье мы используем существующий контроллер (Intel 82527) для иллюстраций приложения анализа. Затем мы применяем тесты SAE для автомобильных систем класса C (устройства управления повышенной надежности) [4]. Затем мы расширяем анализ CAN для рассмотрения некоторых вопросов отказоустойчивости.

Статья построена так: следующий раздел описывает поведение CAN (на примере Intel 82527) и используемую модель системы. Раздел 3 применяет базовый анализ распределения времени процессора к Intel 82527. Раздел 4, затем, применяет этот анализ к стандартным тестам, используя несколько методов. Раздел 5 расширяет основной анализ для рассмотрения восстановления при сбоях в CAN, и показывает как пересмотренный анализ может быть вновь применен к стандартным тестам, уже при наличии определенных требований к отказоустойчивости. Наконец, в разделе 6 обсуждаются некоторые нестандартные вопросы, и даются рекомендации.

2.Модель системы

CAN - шина широковещательной рассылки, к которой через интерфейс подключаются несколько процессоров (Рисунок 1).

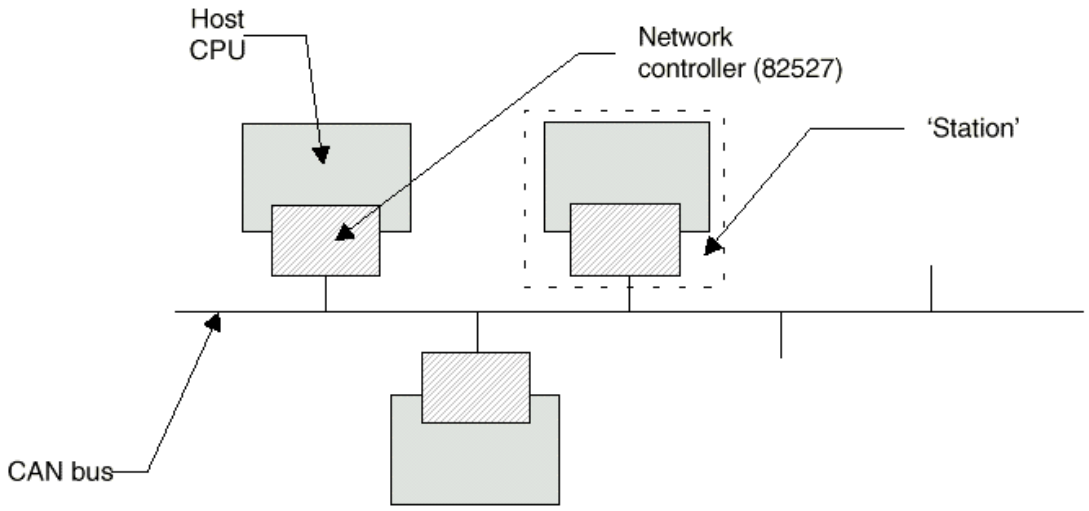

Рисунок 1. Архитектура CAN.

Источник данных передает сообщение, содержащее от 1 до 8 ('октеты') байтов. Сообщение может передаваться периодически, апериодически или по требованию. Так, например, некоторая информация, например 'скорость движения', может быть закодирована как 1 байтовое сообщение и передаваться каждые 100 миллисекунд. Источнику данных назначен уникальный идентификатор, представленный 11 битовым номером (всего возможно 2032 идентификатора - CAN запрещает идентификаторы с семью наиболее значимыми битами равными '1'). Идентификаторы служат для двух целей: фильтрация сообщений при приеме и назначение приоритетов сообщениям. Поскольку мы имеем дело с закрытыми управляющими системами, у нас есть фиксированный набор сообщений, каждое имеющее уникальные приоритет и временные характеристики, как, например, скорость передачи и (максимальный) размер.

Станция в шине CAN может получить сообщение по идентификатору сообщения: если некоторому процессору нужно получить скорость движения автомобиля(например), он посылает идентификатор контроллеру шины. Только сообщения с желаемыми идентификаторами принимаются и передаются в этот процессор. Таким образом сообщение в CAN не имеет конкретного адресата.

Использование идентификатора как приоритета является наиболее важной чертой CAN в смысле производительности в реальном времени. Для любой архитектуры шины должен быть способ обрабатывать конкурирующие запросы: в шине TDMA каждой станции предоставляется определенный период времени, когда она может передавать. В стандарте Ethernet каждая станция ждет молчания в кабеле и затем начинает передачу. Если более чем одна станция пытается передавать одновременно, они все обнаруживают это, ждут период времени случайной длительности и пытаются передавать снова в следующий раз, когда кабель освободится. Ethernet - пример шины широковещательной рассылки с опросом несущей , поскольку каждая станция ожидает пока шина не освободится (то есть отсутствует несущая), и проверяет свой собственный трафик для обнаружения коллизий. CAN - также шина широковещательной рассылки с опросом несущей, но использует более систематический подход при обработке конкуренции. Поле идентификатора в сообщении CAN используется для управления доступом к шине после коллизий, используя преимущества определенных электрических характеристик шины.

В CAN, если несколько станций передают параллельно и одна станция передает '0', то все

станции, слушающие шину увидят '0'. И наоборот, только если все станции передают '1', все процессоры, слушающие шину, увидят '1'. В терминологии CAN, бит '0' бит называется доминантным, а '1' - рецессивным. Фактически, шина CAN действует подобно большому элементу "И", где каждая станция видит выходной сигнал с него. Этот механизм используется для разрешения коллизий: каждая станция ожидает освобождения шины (как в Ethernet). Когда обнаружено молчание, каждая станция начинает передавать самое срочное сообщение, содержащееся в ее очереди, прослушивая при этом шину. Сообщение кодируется так, чтобы наиболее значимый бит поля идентификатора передавался первым. Если станция передает рецессивный бит идентификатора сообщения, но, прослушивая шину, видит доминантный бит, считается что обнаружена коллизия. Станция знает тогда, что сообщение, которое она передает является не самым важным сообщением в системе, прекращает передачу и ждет освобождения шины. Если станция передает рецессивный бит и видит рецессивный бит в шине то, возможно, она передает самое срочное сообщение, и начинает передачу следующего бита поля идентификатора. Поскольку CAN требует уникальности идентификаторов в пределах системы, станция, передающяя последний (наименее значимый) бит идентификатора, не обнаружив коллизию, очевидно, передает сообщение с самым верхним приоритетом, и следовательно может начать передавать тело сообщения (если идентификаторы не были бы уникальными, попытка двух станций передать различные сообщения с одинаковыми идентификаторами вызвала бы коллизию после того, как арбитражный процесс завершился, и произошла бы ошибка передачи).

Необходимо сделать несколько общих замечаний относительно этого арбитражного протокола. Во-первых, сообщение с меньшей величиной идентификатора имеет более высокий приоритет. Во-вторых, самое срочное сообщение проходит арбитражный процесс без помех, в то время как все остальные станции прекратили передачу и ждут освобождения шины. Все тело сообщения таким образом передается без прерываний.

Накладные расходы на передачу одного кадра в CAN в общей сложности составляют 47 битов (включая 11 битов для поля идентификатора, 4 бита для поля длины сообщения, 16 битов для поля CRC, 7 битов для сигнала конца кадра, и 3 бита для перерыва между кадрами). Некоторые из этих полей 'дополнены битами': когда посылаются пять последовательных битов одинаковой полярности, контроллер вставляет в поток дополнительный бит противоположной полярности (это дополнение битами используется как часть механизма контроля ошибок). Из 47 системных битов кадра, 34 подвергаются битовому дополнению. Поле данных в сообщении (от 0 до 8 байтов) также подвергается битовому дополнению. Минимальное сообщение в CAN - 47 битов, самое большое - 130 битов.

У CAN есть еще несколько особенностей, наиболее важные из которых - протокол восстановления при сбоях, и сообщения 'дистанционного запроса передачи'. CAN может выполнять несколько проверок на наличие ошибки, включая использование 16 битового CRC (примененный всего к 8 байтами данных CRC-16 обеспечивает высокую вероятность обнаружения ошибки), нарушение правила 'битового дополнения', и отсутствие бита подтверждения от станций-получателей. Когда любая станция (включая передающую) обнаруживает ошибку, она немедленно сигнализирует об этом, передавая кадр ошибки. Он состоит из шести битов одинаковой полярности, что заставляет все остальные станции также обнаружить ошибку. Эти станции параллельно передают кадры ошибки, после чего все станции вновь синхронизируются. Большинство контроллеров автоматически снова входят в арбитражный процесс, чтобы вновь передать неудашееся сообщение. Для восстановления после ошибки по протоколу требуется передать не более 29 битов кадра

ошибки, и повторно послать неудавшееся сообщение.

Наряду с сообщениями данных, существуют также сообщения типа 'дистанционный запрос передачи'(remote transmission request - RTR). Эти сообщения являются бессодержательными (то есть - нулевой длины) и имеют специальное назначение: они обязывают станцию, удерживающую сообщение данных с тем же идентификатором, передать это сообщение. Сообщения RTR предназначены для быстрого получения редко использующихся данных. Тем не менее, стандартный тест [4] не требует сообщений RTR, и эти типы сообщений далее не обсуждаются .

Анализ может быть использован только в применении к конкретному контроллеру, поскольку различные контроллеры могут иметь различные характеристики (контроллер Philips 82C200 может иметь значительно худшие временные показатели,чем 'идеальный' [6]). Контроллер подключается к процессору через двухпортовое RAM (DPRAM), посредством чего процессор и контроллер могут иметь доступ к памяти одновременно. Идеальный контроллер CAN должен содержать 2032 слота (области в памяти) для сообщений, и сообщение для посылки просто копируется в слот, соответствующий идентификатору сообщения. Полученное сообщение должно также копироваться в соответствующий слот. Однако, поскольку слот потребовал бы по крайней мере 8 байтов, всего потребовалось бы по меньшей мере 16526 байтов памяти. В рассматриваемой среде такое количество памяти было бы недопустимо дорого, и в Intel 82527 реализовано компромисное решение - всего 15 слотов. Один из этих слотов посвящен получению сообщений; остальные 14 сегментов могут устанавливаться на передачу или на прием сообщений. Каждый сегмент может отображаться на любой идентификатор; сегменты запрограммированные для получения, могут устанавливаться для получения любого сообщения, соответствующего маске идентификатора. Каждый слот может независимо программироваться для генерации прерывания при получении сообщения в слот, или, при посылке сообщения из слота. Это позволяет протоколы 'рукопожатия' с процессором, разрешая данному слоту мультиплексироваться на несколько сообщений. Это важно для управления выделенным приемным слотом 15: этот специальный слот подвергается 'двойной буферизации' так, что процессор может очистить один буфер пока второй буфер находится в распоряжении контроллера. В этой статье мы предполагаем, что слоты статически распределяются для сообщений, при этом слот 15 используется для получения сообщений, для которых не хватает остальных слотов. У 82527 есть особенность, заключающаяся в том, что сообщения загруженные в слоты входят в арбитражнай процесс в порядке номеров слотов, а не в порядке идентификаторов (и следовательно приоритетов). Поэтому важно распределять сообщения в слотах в правильном порядке.

Теперь мы опишем 'модель системы' для передачи сообщений, которую мы можем проанализировать. Считается, что система состоит из статического набора срочных сообщений в реальном времени, каждое статически закреплено за несколькими станциями, подключенными к шине. Эти срочные сообщения - как правило управляющие сообщения, и имеют определенные сроки, в которые они должны быть доставлены, иначе может произойти серьезный сбой в работе всей системы. Сообщения, как правило, ставятся в очередь задачей, работающей на процессоре (термином 'задача' охватывается несколько видов деятельности процессора, от обработчиков прерываний до сложных процессов, выполняемых операционной системой). Данная задача должна вызываться некоторым событием, и тратить какое-то время, чтобы поставить сообщение в очередь. Поскольку это время не фиксировано заранее, есть некоторый разброс между последующими постановками сообщений очереди; мы называем это разбросом очереди (queuing jitter). Для целей данной статьи, мы допускаем что время между вызовами данной

задачи минимально; это время называется периодом (конечно, задача могла бы быть запущена только один раз, возможно, в качестве реакции на аварийную ситуацию, и, поэтому, имела бы бесконечный период). Если данная задача посылает сообщение не более одного раза при нескольких последовательных ее вызовах, то говорят что, сообщение наследует период у задачи.

Данному сообщению присваевается фиксированный идентификатор (и следовательно фиксированный приоритет). Мы предполагаем, что каждое данное срочное сообщение должно быть заданного размера (т. е. содержать определенное количество байтов). Имея определенный размер, и определенную скорость передачи, мы можем определить максимальную нагрузку на шину, и можем применить анализ планирования для получения времени доставки для каждого сообщения.

Мы предполагаем, что может также существовать некоторое колическво несрочных (soft) сообщений в реальном времени: эти сообщения не имеют жесткого срока доставки, и могут теряться при передаче (например, процессор - адресат может быть слишком занятым, чтобы получать их). Они посылаются как 'добавка' системе (т. е. если они прибывают в нужное время, то некоторый аспект качества системы улучшается). В этой работе мы не обсуждаем специальные алгоритмы их посылки , и для простоты будем считать, что они посылаются в "фононовом режиме" (т. е. они имеют приоритет ниже, чем у всех срочных сообщений) (Существует несколько алгоритмов, которые могут в принципе привести к очень короткому среднему времени ожидания для несрочных сообщений; применение этих алгоритмов в шине CAN исследуется в Йорке)

Как было упомянуто раньше, постановка в очередь срочных сообщений может происходить со случайным разбросом во времени. Следующая диаграмма иллюстрирует это:

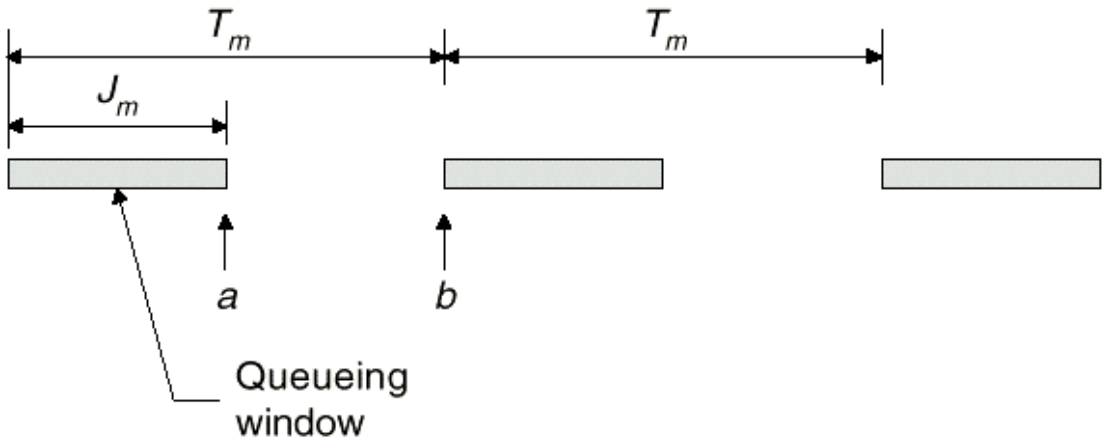

Рисунок 2. Разброс в очереди сообщений.

Темные прямоугольники на данной диаграмме представляют собой 'окна', когда задача на процессоре может поставить сообщение в очередь. Изучение разброса важно поскольку игнорирование его привело бы к не совсем верным результатам анализа. Например, игнорируя разброс на рисунке 2, можно прийти к предположению, что сообщение m могло бы быть поставлено в очередь самое большее один раз в интервал длительности (b - a). На самом же деле, в интервале (a .. b) сообщение могло бы быть поставлено в очередь дважды: один раз в a (как можно позже в первом 'окне' постановки в очередь), и один раз в

b (как можно раньше в следующем 'окне'). Разброс при постановке в очередь может быть определена как разница между самым ранним и самым поздним возможными временами постановки данного сообщения в очередь. В действительности, возможно уменьшить разброс организации очереди если мы знаем в каком месте при выполнении задача ставит сообщение в очередь (например, перед постановкой сообщения в очередь задаче необходимо выполнить небольшое количество вычислений, и следовательно событие b на диаграммах произойдет после старта задачи); подробнее об этом см. [13].

Диаграмма выше также показывает, как период сообщения может зависеть от задачи, посылающей сообщение. Например, если сообщение посылается один раз за каждый вызов задачи, то сообщение наследует период равный периоду задания.

Для того, чтобы уменьшить разброс очереди, мы могли бы разложить задачу, генерирующую сообщение, на две задачи: первая задача вычисляет содержимое сообщения, а вторая - просто ставит это сообщение в очередь. Вторая задача вызывается через определенное время после первой, так что первая всегда успеет завершиться прежде, чем начнется вторая. Поскольку у второй задачи очень немного работы, она будет исполняться достаточно быстро, и разброс очереди унаследованный сообщением будет следовательно небольшой (эта техника обсуждается более подробно в [14]).

Оговорим вкратце как сообщение обрабатывается при получении. В станции-адресате результаты пришедшего сообщения должны быть сделаны доступными. Если сообщение - спорадическое (т.е. посланное как результат случайного события), то должна быть программа, которая запускается с прибытием сообщения. В этом случае, прибытие сообщения должно вызвать прерывание в процессоре (и следовательно попасть в слот 15 в контроллере шины Intel 82527). Конечно, некоторая задача может ожидать этого сообщения и опрашивать шину, но если время его прохождения через шину небольшое, то по сравнению с ним период опроса может быть неприемлемо длинным. С приемом периодического сообщения можно справиться не прибегая к генерации прерывания: сообщение может быть статически присвоено слоту в 82527 и затем извлекаться оттуда прикладной программой. Эта программа могла бы вызываться по таймеру, и потому была бы гарантия прибытия сообщения во время ее исполнения.

Как видно, единственное требование, накладываемое на коммуникационную шину - это заданное время доставки сообщения. Теперь перейдем к их расчету.. Несомненно, этот анализ составит ключевую часть более общего анализа всей системы для вычисления гарантий на временные параметры; подобного рода полный анализ исследуется в Йорке.

Описав базовую модель для CAN и всей системы, мы можем проанализировать поведение конкретного срочного сообщения.

3. Анализ 82527 CAN

В этом разделе мы представляем анализ, который позволяет ограничить неблагоприятное время ожидания данного сообщения в реальном времени. В своем анализе мы опирались на теорию [7, 8, 9]. Однако, есть некоторые предположения сделанные при данном анализе. Во-первых, deadline данного сообщения m (обозначенное D m) не должен быть более чем период сообщения (обозначенного T m). Во-вторых, контроллер шины не должен пропускать менее приоритетные сообщения, если есть "висящее" сообщение с более высоким приоритетом (то есть контроллер не может освободить шину между

посылкой одного сообщения и вводом любого сообщения в произвольный момент; имейте в виду, что контроллер Philips 82C200 CAN не отвечает этому предположению).

Неблагоприятное время ответа данного сообщения m обозначается R m , и определяется как самое длинное время между началом организации очереди задания m и самым последним моментом прихода сообщения. Имейте в виду, что это время включает время необходимое для выполнения и постановки в очередь сообщения m, и это на первый взгляд странное определение (измерение времени от постановки сообщения в очередь до момента прибытия могла показаться лучше). Тем не менее, содержимое сообщения отражает результаты некоторого действия предпринятого заданием (само запущенное в ответ на некоторое событие), и более желательно измерить время от и до, связанное с данным событием.

Неустойчивость данного сообщения m обозначена J m , и она берется из времени ответа задания на главном процессоре. Если эти задания планируются фиксированным приоритетом pre-emptive scheduling, тогда данная работа может ограничить время постановки сообщения в очередь [10, 7] и следовательно определять неустойчивость очереди.

Мы упомянули раньше, как CAN обслуживает алгоритмы с фиксированным приоритетом. Однако, сообщение не всегда полностью pre-emptive, поскольку

срочное (с высоким приоритетом) сообщение не может прервать сообщение, которое уже передается 4. Работа Berns [9] принимает это во внимание, и из другой работы посвященной этой теме [8] мы получаем формулу, которая ограничивает неблагоприятное время данного сообщения в реальном времени следующим выражением:

$$R_m = J_m + w_m + C_m \qquad (1)$$

J m - неустойчивость очередности сообщения m, и она дает самое последнее время посылки сообщения в очередь, относительно начала посылки задания. w m представляет неблагоприятную задержку посылки в очередь сообщения m (как из-за как высших по приоритету сообщений pre-empting, так и из-за более низких по приоритету сообщений, который уже заняли шину).

C m представляет собой самое длинное время физической посылки сообщения m в шину. Это время включает время взятое накладными расходами фрейма, данными, и дополнительными битами (возвращаясь к главе 2, где сообщение содержало кроме данных 34 бита накладных расходов и 5 дополнительных битов). Следующее уравнение дает C m:

$$C_m = \left(\left\lfloor \frac{34 + 8s_m}{5} \right\rfloor + 47 + 8s_m \right) \tau_{bit}$$

s m обозначает размер сообщения m в байтах. t является тактом передачи шины (в шине, работающей в 1 Mbit/сек это - 1ms).

Задержка организации очереди дается:

$$w_m = B_m + \sum_{\forall j \in hp(m)} \left\lceil \frac{w_m + J_j + \tau_{bit}}{T_j} \right\rceil C_j \quad (2)$$

Множество hp(m) является множеством сообщений в системе с более высоким нежели у m приоритетом. T j - период данного сообщения j, и J j - неустойчивость очередности сообщения. B m - самое длинное время которым данное сообщение m может задерживаться более низкими по приоритету сообщениями (это равняется времени взятому, чтобы передать самое большое более низкоприоритетное сообщение), и оно определяется:

$$B_m = \max_{\forall k \in lp(m)} (C_k)$$

Где lp(m) - множество более низких по приоритету сообщений. Заметим, что если есть неограниченный набор сообщений неопределенного размера в реальном времени, то B m равняется 130t битам.

Заметим также, что в уравнении 2 w m появляется как в левой так и в правой части, и следовательно уравнение не разрешимо относительно w m. Простое решение получается если использовать рекуррентное соотношение:

$$w_m^{n+1} = B_m + \sum_{\forall j \in hp(m)} \left\lceil \frac{w_m^n + J_j + \tau_{bit}}{T_j} \right\rceil C_j$$

Положим w m 0 равное нулю. Итерации продолжаются до полной стационарной сходимости

$$(i.e.\ w_m^{n+1} = w_m^n).$$

Вышеуказанные уравнения не знают ничего о том как выбираются идентификаторы и, следовательно, приоритеты. Тем не менее, из работ [11, 8] мы знаем, что оптимальным упорядочением приоритетов будет является deadline monotonic: заданию с малой величиной D - J должен быть назначен более высокий приоритет.

Теперь мы можем непосредственно приступить к анализу тестов SAE benchmark [4].

4. SAE benchmark

SAE описывает набор сигналов посланных между семью различными крупными элементами системы в прототипе электрического автомобиля. Хотя автомобильная управляющая система была создана используя сканирующие связи, набор таких сигналов предоставляет хорошую возможность проиллюстрировать применение шины CAN в сложных распределенных системы управления реального времени.

Семь крупных элементов системы это: батареи ('Battery'), контроллер двигателя ('V/C'), контроллер инвертора/двигателя ('I/M C'), дисплей панели инструментов ('Ins'), управление водителя ('Drive'), тормоза ('Brakes'), и управление трансмиссией ('Trans'). Сеть, соединяющая эти элементы системы, должна оперировать в общей ложности 53

сообщениями, некоторые из которых содержат спорадические сигналы, а некоторые шлют управляющую информацию периодически. Периодическое оообщение имеет фиксированный период, и требует время ожидания менее или равное этому периоду. Спорадические сообщения требуют различное время ожидания связанное с конкретной ситуацией: например, все сообщения посланные в результате действий водителя требуют времени ожидания 20ms так как ответ водителю должен быть немедленным.

Отсылаем читателя к работе Копетца (Kopetz [3]) для более подробного описания benckmark. Земетим, что Копетц усиливает интерпретацию контрольной спецификации, требуя значительно малых времен синхронизации там, где стандарт отказывается определять их (например, требование времени ожидания 20ms для сообщений поступающих от водителя - требование навязанное Копетцом, а не стандарт). Есть еще одна некоторая неопределенность в benchmark: модель системы, принятая в этой статье, требует, чтобы даже спорадическим сообщениям имели некий период (максимальной время, в течение которого они могут произойти), но никакие периоды для спорадических сообщений в реальной ситуации не могут быть введены (Копетц допускает, что спорадические сообщения имеют период 20ms). Подобно Копетцу, мы уславливаемся, что все величины принимают разумные значения. Мы также предполагаем неустойчивость организации очереди величин.

Следующая таблица подробно описывает требования к сообщений, которые должны выполняться. Есть в общей сложности 53 сообщения, некоторые из них простые периодические сообщения, а некоторые - случайные сообщения (то есть поставленное в очередь спорадически в ответ на внешнее событие).

Signal	Signal	Size	J	T	Periodic	D	From	To
Number	Description	/bits	/ms	/ms	/Sporadic	/ms		
1	Traction Battery Voltage	8	0.6	100.0	P	100.0	Battery	V/ C
2	Traction Battery Current	8	0.7	100.0	P	100.0	Battery	V/ C
3	Traction Battery Temp, Average	8	1.0	1 100.0	P	1000.0	Battery	V/ C
4	Auxiliary Battery Voltage	8	0.8	100.0	P	100.0	Battery	V/ C
5	Traction Battery Temp, Max.	8	1.1	1000.0	P	1000.0	Battery	V/ C
6	Auxiliary Battery Current	8	0.9	100.0	P	100.0	Battery	V/ C
7	Accelerator Position	8	0. 1	5. 0	P	5.0	Driver	V/ C
8	Brake Pressure, Master Cylinder	8	0.1	5.0	P	5.0	Brakes	V/ C
9	Brake Pressure, Line	8	0.2	5.0	P	5.0	Brakes	V/ C
10	Transaxle Lubrication Pressure	8	0.2	100.0	P	100.0	Trans	V/ C
11	Transaction Clutch Line Pressure	8	0.1	5.0	P	5.0	Trans	V/ C

12	Vehicle Speed	8	0. 4	100.0	P	100.0	Brakes	V/ C
13	Traction Battery Ground Fault	1	1.2	1000.0	P	1000.0	Battery	V/ C
14	Hi& Lo Contactor Open/ Close	4	0.1	50.0	S	5.0	Battery	V/ C
15	Key Switch Run	1	0. 2	50.0	S	20.0	Driver	V/ C
16	Key Switch Start	1	0.3	50.0	S	20.0	Driver	V/ C
17	Accelerator Switch	2	0.4	50.0	S	20.0	Driver	V/ C
18	Brake Switch	1	0.3	20.0	S	20.0	Brakes	V/ C
19	Emergency Brake	1	0.5	50.0	S	20.0	Driver	V/ C
20	Shift Lever (PRNDL)	3	0. 6	50.0	S	20.0	Driver	V/ C
21	Motor/ Trans Over Temperature	2	0.3	1000.0	P	1000.0	Trans	V/ C
22	Speed Control	3	0.7	50.0	S	20.0	Driver	V/ C
23	12V Power Ack Vehicle Control	1	0.2	50.0	S	20.0	Battery	V/ C
24	12V Power Ack Inverter	1	0.3	50.0	S	20.0	Battery	V/ C
25	12V Power Ack I/ M Contr.	1	0.4	50.0	S	20.0	Battery	V/ C
26	Brake Mode (Parallel/ Split)	1	0.8	50.0	S	20.0	Driver	V/ C
27	SOC Reset	1	0. 9	50.0	S	20.0	Driver	V/ C
28	Interlock	1	0.5	50.0	S	20.0	Battery	V/ C
29	High Contactor Control	8	0.3	10.0	P	10.0	V/ C	Battery
30	Low Contactor Control	8	0.4	10.0	P	10.0	V/ C	Battery
31	Reverse and 2nd Gear Clutches	2	0.5	50.0	S	20.0	V/ C	Trans
32	Clutch Pressure Control	8	0.1	5.0	P	5.0	V/ C	Battery
33	DC/ DC Converter	1	1.6	1000.0	P	1000.0	V/ C	Battery
34	DC/ DC Converter Current Control	8	0.6	50.0	S	20.0	V/ C	Battery
35	12V Power Relay	1	0. 7	50.0	S	20.0	V/ C	Battery
36	Traction Battery Ground Fault Test	2	1.7	1000.0	P	1000.0	V/ C	Brakes

37	Brake Solenoid	1	0.8	50.0	S	20.0	V/ C	Brakes
38	Backup Alarm	1	0.9	50.0	S	20.0	V/ C	Brakes
39	Warning Lights	7	1.0	50.0	S	20.0	V/ C	Ins.
40	Key Switch	1	1. 1	50.0	S	20.0	V/ C	I/ M C
41	Main Contactor Close	1	0.3	50.0	S	20.0	I/ M C	V/ C
42	Torque Command	8	0.2	5.0	P	5.0	V/ C	I/ M C
43	Torque Measured	8	0. 1	5. 0	P	5.0	I/ M C	V/ C
44	FWD/ REV	1	1. 2	50.0	S	20.0	V/ C	I/ M C
45	FWD/ REV Ack.	1	0. 4	50.0	S	20.0	I/ M C	V/ C
46	Idle	1	1.3	50.0	S	20.0	V/ C	I/ M C
47	Inhibit	1	0.5	50.0	S	20.0	I/ M C	V/ C
48	Shift in Progress	1	1.4	50.0	S	20.0	V/ C	I/ M C
49	Processed Motor Speed	8	0. 2	5. 0	P	5.0	I/ M C	V/ C
50	Inverter Temperature Status	2	0.6	50.0	S	20.0	I/ M C	V/ C
51	Shutdown	1	0.7	50.0	S	20.0	I/ M C	V/ C
52	Status/ Malfunction (TBD)	8	0.8	50.0	S	20.0	I/ M C	V/ C
53	Main Contactor Acknowledge	1	1.5	50.0	S	20.0	V/ C	I/ M C

Простейшая попытка в осуществлении задачи на CAN это отобразить каждое из этих сообщений в CAN-сообщение. Спорадические сообщения обычно требуют время ожидания 20 мс или менее (хотя Kopetz дает необходимое время ожидания 5ms для одного из спорадических сообщений). Эти сообщения могут быть поставлены в очередь редко (например, разумно допустить, что там по крайней мере 50 мс проходит между нажатиями педали тормоза). Benchmark не дает периоды для этих сообщений, так что мы принимаем период 50ms для всех спорадических сообщений.

Мы упоминали в разделе 2 как специальное 'выходное задание' могло бы создаваться для каждого сообщения вместе с работой по организации очереди pre-assembled сообщений, и мы считаем, что эта модель принимается для контрольной системы проанализированной здесь. Следующая таблица включает сообщения в порядке приоритета (то есть в D-J порядке), и дает неблагоприятное время ожидания, вычисленное в свете предшествующего раздела. Жирным шрифтом выделены спорадические сигналы. Символ ?? указывает, что сообщение не удовлетворяет своим требованиям по времени ожидания (то есть не гарантируется, что сообщение всегда достигает искомого места в пределах необходимого времени); символ '-' указывает, что время ответа CAN может определяться неверно, поскольку не гарантируется, что сообщение бедет послано до того, как следующее будет поставлено в очередь (то есть R > D - J).

Signal N.o	Size /bytes	J /ms	T /ms	D /ms	R (125Kbit/s)	R (250Kbit/s)	R (500Kbit/s)	R (1Mbit/ s)
14	1	0.1	50.0	5.0	1.544	0.772	0.386	0.193
9	1	0.2	5.0	5.0	2.048	1.024	0.512	0.256
49	1	0.2	5.0	5.0	2.552	1.276	0.638	0.319
42	1	0.2	5.0	5.0	3.056	1.528	0.764	0.382
8	1	0.1	5.0	5.0	3.560	1.780	0.890	0.445
7	1	0.1	5.0	5.0	4.064	2.032	1.016	0.508
43	1	0.1	5.0	5.0	4.568	2.284	1.142	0.571
11	1	0.1	5.0	5.0	_ 5.072	2.536	1.268	0.634
32	1	0.1	5.0	5.0	_ -	2.788	1.394	0.697
29	1	0.3	10.0	10.0	_ 10.112	3.040	1.520	0.760
30	1	0.4	10.0	10.0	_ -	3.292	1.646	0.823
53	1	1.5	50.0	20.0	_ 25.232	3.544	1.772	0.886
48	1	1.4	50.0	20.0	_ 29.768	3.796	1.898	0.949
46	1	1.3	50.0	20.0	_ 39.344	4.048	2.024	1.012
44	1	1.2	50.0	20.0	_ 39.848	4.300	2.150	1.075
40	1	1.1	50.0	20.0	_ -	4.552	2.276	1.138
39	1	1.0	50.0	20.0	_ -	4.804	2.402	1.201
27	1	0.9	50.0	20.0	_ -	7.072	2.528	1.264
38	1	0.9	50.0	20.0	_ -	7.324	2.654	1.327
37	1	0.8	50.0	20.0	_ -	7.576	2.780	1.390
52	1	0.8	50.0	20.0	_ -	7.828	2.906	1.453
26	1	0.8	50.0	20.0	_ -	8.080	3.032	1.516
35	1	0.7	50.0	20.0	_ -	8.332	3.158	1.579
51	1	0.7	50.0	20.0	_ -	8.584	3.284	1.642
22	1	0.7	50.0	20.0	_ -	8.836	3.410	1.705

34	1	0.6	50.0	20.0	_ -	9.088	3.536	1.768
20	1	0.6	50.0	20.0	_ -	9.340	3.662	1.831
50	1	0.6	50.0	20.0	_ -	9.592	3.788	1.894
31	1	0.5	50.0	20.0	_ -	9.844	3.914	1.957
47	1	0.5	50.0	20.0	_ -	12.616	4.040	2.020
28	1	0.5	50.0	20.0	_ -	12.868	4.166	2.083
19	1	0.5	50.0	20.0	_ -	13.120	4.292	2.146
25	1	0.4	50.0	20.0	_ -	13.372	4.418	2.209
17	1	0.4	50.0	20.0	_ -	13.624	4.544	2.272
45	1	0.4	50.0	20.0	_ -	13.876	4.670	2.335
24	1	0.3	50.0	20.0	_ -	14.128	4.796	2.398
16	1	0.3	50.0	20.0	_ -	14.380	4.922	2.461
18	1	0.3	50.0	20.0	_ -	14.632	6.056	2.524
41	1	0.3	50.0	20.0	_ -	14.884	6.182	2.587
23	1	0.2	50.0	20.0	_ -	17.152	6.308	2.650
15	1	0.2	50.0	20.0	_ -	17.404	6.434	2.713
6	1	0.9	100.0	100.0	_ -	17.656	6.560	2.776
4	1	0.8	100.0	100.0	_ -	17.908	6.686	2.839
2	1	0.7	100.0	100.0	_ -	18.160	6.812	2.902
1	1	0.6	100.0	100.0	_ -	18.412	6.938	2.965
12	1	0.4	100.0	100.0	_ -	18.664	7.064	3.028
10	1	0.2	100.0	100.0	_ -	18.916	7.190	3.091
36	1	1.7	1000.0	1000.0	_ -	19.168	7.316	3.154
33	1	1.6	1000.0	1000.0	_ -	19.420	7.442	3.217
13	1	1.2	1000.0	1000.0	_ -	19.672	7.568	3.280
5	1	1.1	1000.0	1000.0	_ -	22.444	7.694	3.343
3	1	1.0	1000.0	1000.0	_ -	22.696	7.820	3.406

21	1	0.3	1000.0	1000.0	-	22.948	7.946	3.469

Есть проблема с методом подхода к преобразованию каждого сигнала в CAN сообщения: V/C подсистема передает более чем 14 типов сообщений, и таким образом 82527 не может быть использован (напомним, что в 82527 есть 15 выводов, один из которых является только входом). Мы вскоре возвратимся к этой проблеме.

Как можно видеть, при скорости шины 125Kbit/s система не может гарантированно удовлетворить всем временным ограничениям. Для того, чтобы увидеть основную причину, рассмотрим следующую таблицу:

Bus Speed	Message Utilisation	Bus Utilisation	A
125 Kbit/ s	15.91%	125.29%	-
250 Kbit/ s	7.95%	62.64%	1.14
500 Kbit/ s	3.98%	31.32%	3.09
1Mbit/ s	1.99%	15.66%	5.79

'Message utilisation' вычислена используя количество байтов данных в данном CAN сообщении. 'Bus utilisation' вычисляется с использованием общего числа битов (включая потери) в данном CAN сообщении. Далее идет столбец озаглавленный 'Details breakdown utilisation' [12] системы для данной скорости шины. 'Breakdown utilisation' представляет собой самое большое время такое, что все сообщения успевают пройти (то есть все требования времени ожидания удовлетворяются). Это дает понятие о том сколько резервов есть в системе: величина близкая к, но больше, чем 1 указывает, что хотя система загружена, есть небольшое место для повышения нагрузки. Символ '-' для скорости шины 125Kbit/s указывает, что никакая величина для 'Breakdown utilisation' не может быть найдена, даже при a = 0 это все еще невозможно. Как можно видеть, есть большое различие между сообщением и 'Bus utilisation'. Дело в том, что CAN сообщение имеет сравнительно большие дополнительные потери. При скорости шины 125Kbit/s 'Bus utilisation' - больше чем 100%, и следовательно не надо удивляться, что система неудовлетворяет требованиям.

Один из путей уменьшения 'Bus utilisation' (и 'Message utilisation') в 'piggyback' сообщениях посланных из того же источника. Например, рассмотрим Battery подсистему: она периодически посылает четыре однобайтовых сообщения, каждое с периодом 100 мс (сообщения номер 1, 2, 4, и 6). Если бы мы собрали их в одно единственное сообщение, тогда мы могли послать одно четырехбайтовое сообщение. Это может и должно уменьшить потери, и следовательно 'Bus utilisation'. Другое преимущество такого способа в том, что количество слотов требующихся в контроллере шины уменьшается (мы имеем только 14 выводов в 82527 CAN контроллере, а V/C подсистема имеет более чем 14 сигналов).

Мы можем сделать 'piggyback' следующие периодические сообщения:

New message name	Size	T	Composed from

	/bytes	/ms	
Battery high rate	4	100.0	1,2,4,6
Battery low rate	3	1000.0	3,5,13
Brakes high rate	2	5.0	8,9
I/ M C high rate	2	5.0	43,49
V/ C high rate	4	5.0	32,42,29,30
V/ C low rate	1	1000.0	33,36

Эта операция должна осуществляться прикладными заданиями, обрабатывающими содержание сообщения как выше; но необходимо, чтобы различные сигналы собранные в одном единственном сообщении были бы однозадачно и просто организованы в очередь сообщений, причем периодически. Поскольку есть и меньшие сообщения, которые посылаются с данного узла, неустойчивость организации очереди данного сообщения должна быть немного меньше.

Заметим, что сигналы 29 и 30 должны быть посланы с периодом 10ms, но, что они объединенные в одно сообщение ('V/C high rate') могут быть посланны один раз каждые 5ms (и таким образом каждое другое посланное сообщение должно содержать информацию об отсутствии данных для этих сигналов). Нам нужно делать так, потому что не более чем 14 типов сообщений могут быть посланны из V/C подсистемы.

Следующая таблица дает детально описывает временные характеристики всех сообщений в этом новом множестве сообщений:

Signal N.o	Size	J	T	D	R	R	R	R
	/bytes	/ms	/ms	/ms	(125Kbit/s)	(250Kbit/s)	(500Kbit/s)	(1Mbit/ s)
14	1	0.1	50.0	5.0	1.544	0.772	0.386	0.193
8,9	2	0.1	5.0	5.0	2.128	1.064	0.532	0.266
7	1	0.1	5.0	5.0	2.632	1.316	0.658	0.329
43,49	2	0.1	5.0	5.0	3.216	1.608	0.804	0.402
11	1	0.1	5.0	5.0	3.720	1.860	0.930	0.465
32,42,29,30	4	0.1	5.0	5.0	4.456	2.228	1.114	0.557
23	1	0.5	50.0	20.0	4.960	2.480	1.240	0.620
24	1	0.4	50.0	20.0	8.376	2.732	1.366	0.683
25	1	0.3	50.0	20.0	8.880	2.984	1.492	0.746
28	1	0.2	50.0	20.0	9.384	3.236	1.618	0.809

18	1	0.2	50.0	20.0	9.888	3.488	1.744	0.872
15	1	0.9	50.0	20.0	10.392	3.740	1.870	0.935
16	1	0.8	50.0	20.0	13.808	3.992	1.996	0.998
17	1	0.7	50.0	20.0	14.312	4.244	2.122	1.061
19	1	0.6	50.0	20.0	14.816	4.496	2.248	1.124
20	1	0.5	50.0	20.0	15.320	4.748	2.374	1.187
22	1	0.4	50.0	20.0	18.736	5.000	2.500	1.250
26	1	0.3	50.0	20.0	19.240	6.708	2.626	1.313
27	1	0.2	50.0	20.0	19.744	6.960	2.752	1.376
41	1	0.7	50.0	20.0	_ 20.248	7.212	2.878	1.439
45	1	0.6	50.0	20.0	_ 23.664	7.464	3.004	1.502
47	1	0.5	50.0	20.0	_ 24.168	7.716	3.130	1.565
50	1	0.4	50.0	20.0	_ 24.672	7.968	3.256	1.628
51	1	0.3	50.0	20.0	_ 25.176	8.220	3.382	1.691
52	1	0.2	50.0	20.0	_ 28.592	8.472	3.508	1.754
31	1	1.2	50.0	20.0	_ 29.096	8.724	3.634	1.817
34	1	1.1	50.0	20.0	_ 29.600	8.976	3.760	1.880
35	1	1.0	50.0	20.0	_ 30.104	9.228	3.886	1.943
37	1	0.9	50.0	20.0	_ 33.520	9.480	4.012	2.006
38	1	0.8	50.0	20.0	_ 34.024	9.732	4.138	2.069
39	1	0.7	50.0	20.0	_ 34.528	9.984	4.264	2.132
40	1	0.6	50.0	20.0	_ 35.032	11.692	4.390	2.195
44	1	0.5	50.0	20.0	_ 38.448	11.944	4.516	2.258
46	1	0.4	50.0	20.0	_ 38.952	12.196	4.642	2.321
48	1	0.3	50.0	20.0	_ 39.456	12.448	4.768	2.384
53	1	0.2	50.0	20.0	_ 39.960	12.700	4.894	2.447
1,2,4,6	4	0.6	100.0	100.0	43.608	13.068	5.078	2.539

12	1	0.3	100.0	100.0	44.112	13.320	5.932	2.602
10	1	0.2	100.0	100.0	44.616	13.572	6.058	2.665
3,5,13	3	0.7	1000.0	1000.0	45.272	13.900	6.222	2.747
21	1	0.3	1000.0	1000.0	48.688	14.152	6.348	2.810
33,36	1	1.3	100.0	1000.0	49.192	14.404	6.474	2.873

Заметим, что требования по времени теперь встают для большего числа сообщений нежели чем для простых сообщений. Мы также можем послать вышеуказанные сообщения используя контроллер Intel 82527 CAN, поскольку никакая подсистема не посылает теперь более чем 14 типов сообщений.

Для сравнения, следующая таблица дает 'Message utilisation' для вышеуказанного множества сообщений, и 'Breakdown utilisation' системы:

Bus	Message	Bus	A
Speed	Utilisation	Utilisation	
125 Kbit/s	15.20%	75.77%	0.26
250 Kbit/s	7.60%	37.89%	1.90
500 Kbit/s	3.80%	18.94%	4.12
1Mbit/s	1.9%	9.47%	7.61

Как можно увидеть, объединение сообщений ведет к уменьшению накладных расходов, и следовательно уменьшение 'Bus utilisation'. В общем, это в свою очередь ведет к возрастающему времени выполнения в реальном времени. Тем не менее, объединения сигналы в одно единственное сообщение мы требуем, чтобы сигналы всегда генерировались одновременно. Это ограничивает применимость такого способа. Возможно объединять сигналы, которые не обязательно сгенерированы вместе (например, спорадические сигналы). Метод, который мы берем, посылает периодически 'серверное' сообщение. Спорадический сигнал, который должен посылаться, загружается в память главного центрального процессора. Когда 'серверное' сообщение послано, происходит опрос, пришли ли какие-либо сигналы, и соответственно заполняется содержимое сообщения. Этим способом спорадический сигнал может задерживаться вплоть до времени опроса плюс неблагоприятное время ожидания 'серверного' сообщения. Так, при объединении нескольких спорадических сигналов с требованиями времени ожидания 20ms или более , сообщение сервера с периодом 10ms и неблагоприятное время ответа 10ms должно быть достаточным. Кроме того, можно например использовать сообщение сервера с периодом 15ms и неблагоприятного времени ответа 5ms.

Мы преобразуем наш набор сообщений, чтобы включить в него сообщения сервера. Мы выбираем сообщения сервера с периодом 10ms и требованием времени ожидания 10ms. Следующая таблица иллюстрирует полученный результат:

Signal N s o	Size /bytes	J /ms	T /ms	D /ms	R (125Kbit/s)	R (250Kbit/s)	R (500Kbit/s)	R (1Mbit/s)
14	1	0.1	50.0	5.0	1.544	0.772	0.386	0.193
8,9	2	0.1	5.0	5.0	2.128	1.064	0.532	0.266
7	1	0.1	5.0	5.0	2.632	1.316	0.658	0.329
43,49	2	0.1	5.0	5.0	3.216	1.608	0.804	0.402
11	1	0.1	5.0	5.0	3.720	1.860	0.930	0.465
32,41	2	0.1	5.0	5.0	4.304	2.152	1.076	0.538
31,34,35,37,38,39,40, 44,46,48,53	6	0.2	10.0	10.0	5.192	2.596	1.298	0.649
23,24,25,28	1	0.2	10.0	10.0	8.456	2.848	1.424	0.712
15,16,17,19,20,2226,27	2	0.2	10.0	10.0	9.040	3.140	1.570	0.785
41,43,45,47,49,50,51,52	3	0.2	10.0	10.0	9.696	3.468	1.734	0.867
18	1	0.2	50.0	20.0	10.200	3.720	1.860	0.930
1,2,4,6	4	0.3	100.0	100.0	19.088	4.088	2.044	1.022
12	1	0.3	100.0	100.0	19.592	4.340	2.170	1.085
10	1	0.2	100.0	100.0	20.096	4.592	2.296	1.148
3,5,13	3	0.4	1000.0	1000.0	28.904	4.920	2.460	1.230
21	1	0.3	1000.0	1000.0	29.408	6.552	2.586	1.293
33,36	1	0.3	1000.0	1000.0	29.912	6.804	2.712	1.356

Есть два спорадических сигнала, которые остаются чисто спорадическими сообщениями: Сигнал 14 имеет слишком малое время задержки, чтобы ждать опроса. Сигнал 18 - единственное спорадическое сообщение посылаемое от подсистемы Brakes, и следовательно оно не может быть объединено с другими спорадическими сигналами.

Следующая таблица дает 'utilisation' и 'Breakdown utilisation' вышеуказанной системы:

Bus Speed	Message Utilisation	Bus Utilisation	A
125 Kbit/ s	18.46%	84.44%	1.011

250 Kbit/ s	9.23%	42.22%	1.981
500 Kbit/ s	4.62%	21.11%	3.812
1Mbit/ s	2.31%	10.55%	7.082

Обратите внимание, что 'Breakdown utilisation' для системы с 125Kbit/s шиной очень близка к 1, и это показывает, что система загружена почти полностью, и вероятно не сможет разместить более ни одного сигнала. Заметим также, что 'Bus utilisation' повысилась, несмотря на то, что систему теперь адекватна на всех четырех скоростях шины. Хотя накладные расходы теперь больше (из-за опроса для спорадических сообщений), максимальная нагрузка в системе снижена. Эти показывает, что 'utilisation' является существенным фактором только для значительных интервалов; в системах реального времени максимальная нагрузка значительно более важна, и может быть независима от общей 'Bus utilisation'.

5. Расширение анализа : восстановление данных при ошибках

Пока мы допусткали, что никакие ошибки не могут произойти в шине. Тем не менее, анализ раздела 3 можно легко распространить для того, чтобы оперировать при наличии ошибок. Уравнение 2 корректируется на:

$$w_m = B_m + \sum_{\forall j \in hp(m)} \left\lceil \frac{w_m + J_j + \tau_{bit}}{T_j} \right\rceil C_j + E_m(w_m + C_m)$$

Где неблагоприятное время ответа дается уравнением 1. E m (t) характеризует верхнюю границу функции восстановления ошибок, и дает ожидаемую верхнюю границу в соответствии с восстановлением ошибок которые могли бы произойти в интервале длительности t.

Эта функция может быть определена также из наблюдения поведения CAN под условии высоких шумов, или формируя статистическую модель. В этой статье мы используем очень простую функцию ошибки для иллюстрации наших результатов:

Пусть n будет числом ошибок, которые могут произойти в произвольном интервале времени (то есть n ошибок произошли одна за другой); пусть T будет остаточным периодом ошибки (то есть после начальных n ошибок , ошибки не могут произойти прежде, чем пройдет время T). Количество ошибок в интервале длительности t связывается следующим соотношением:

$$n_{error} + \left\lceil \frac{t}{T_{error}} \right\rceil - 1$$

В протоколе CAN, каждая ошибка может вызвать самое большее 29 битов потерь при восстановлении данных, следующих за вновь переданным сообщением. Только сообщения более высокого приоритета, чем сообщение m и само сообщение m может заново передавать и задерживать сообщение m (станция пытаясь вновь передать более низкое по приоритету сообщение после того, как сообщение m поставлено в очередь,

потеряет много времени). Самое большое из этих сообщений:

$$\max_{\forall k \in hp(m) \cup \{m\}} (C_k)$$

Следовательно, ограничение на накладные расходы в интервале, и следовательно на функцию ошибки, дается:

$$E_m(t) = \left(n_{error} + \left\lceil \frac{t}{T_{error}} \right\rceil - 1 \right) \left(29 + \max_{\forall k \in hp(m) \cup \{m\}} (C_k) \right)$$

Мы можем теперь возвратиться к SAE benchmark и увидеть как CAN работает при определенных условиях ошибок. Пусть n равно 4, и пусть T будет 10ms (это очень пессимистичное предположение: при скорости передачи информации 1 Mbit/s это эквивалентно частоте появления ошибок 1 бит в 10 000; размеры CAN предполагают типичную ошибку 1 бит в 10^5).

Следующая таблица дает результаты скорректированного анализа для уже разобранных выше сообщений:

Signal N s o	Size /bytes	J /ms	T /ms	D /ms	R (125Kbit/s)	R (250Kbit/s)	R (500Kbit/s)	R (1Mbit/s)
14	1	0.1	50.0	5.0	3.676	1.896	1.006	0.561
8,9	2	0.1	5.0	5.0	4.580	2.348	1.232	0.674
7	1	0.1	5.0	5.0	_	2.600	1.358	0.737
43,49	2	0.1	5.0	5.0	_	2.892	1.504	0.810
11	1	0.1	5.0	5.0	_	3.144	1.630	0.873
32,41	2	0.1	5.0	5.0	_	3.436	1.776	0.946
31,34,35,37,38,39,40, 44,46,48,53	6	0.2	10.0	10.0	_	4.488	2.302	1.209
23,24,25,28	1	0.2	10.0	10.0	_	4.740	2.428	1.272
15,16,17,19,20,2226,27	2	0.2	10.0	10.0	_	5.032	2.574	1.345
41,43,45,47,49,50,51,52	3	0.2	10.0	10.0	_	6.740	2.738	1.427
18	1	0.2	50.0	20.0	_	6.992	2.864	1.490
1,2,4,6	4	0.3	100.0	100.0	59.949	7.360	3.048	1.582

12	1	0.3	100.0	100.0	69.522	7.612	3.174	1.645
10	1	0.2	100.0	100.0	70.026	7.864	3.300	1.708
3,5,13	3	0.4	1000.0	1000.0	78.834	8.192	3.464	1.790
21	1	0.3	1000.0	1000.0	80.255	8.444	3.590	1.853
33,36	1	0.3	1000.0	1000.0	88.911	8.696	3.716	1.916

Результаты показывают, как CAN сеть работающая на 125Kbit/s не может выдержать предполагаемую частоту появления ошибок. При скорости 250Kbit/s или более система CAN уже может выдержать такую частоту появления ошибок. Заметим, что нам не нужно передавать каждое сообщение несколько раз, и мы можем взамен полагаться на CAN протокол, чтобы обнаружить ошибки и передать повторно неудачные сообщения.

Альтернативная стратегия защиты от ошибок выключает CAN механизм восстановления при ошибках и использует дублирующую зеркальную шину и сообщение посылается в обе шины параллельно (этот метод взят Копетцом из протокола TTP). Тем не менее, мы можем видеть, что это решение значительно менее гибкое нежели использование динамического метода восстановления при ошибках. Сделав предположения о частоте ошибок (на самом деле, для любой системы мы должны сделать такие предположения), затем мы можем использовать эти предположения в динамическом методе восстановления.

6. Выводы

Анализ приведенный в этой статье позволяет сказать, что CAN протокол может использоваться в большом количестве систем работающих в реальном времени. На самом деле использование общесистемных приоритетов для заказа передачи сообщения делает эту систему идеальной управляющей сетью. Использование глобального метода в приоритетах также имеет преимущество, так как существует масса материала, разработанного для фиксированного приоритетного планирования процессора, который легко приспособить для использования CAN. Существуют инструментальные средства, которые включают планирование процессора. Аналогичные инструментальные средства могли бы быть разработаны для CAN. Они должны не только точно предсказать неблагоприятное время ожидания сообщения (для всех классов сообщений в системе), но могли бы также использоваться инженером систем, чтобы проконтролировать интересующее его приложение.

Прилагая анализ в рамках уже существующего benchmark были сделаны оценки применимости. Тем не менее, benchmark не иллюстрирует все преимущества и гибкость CAN, которая обеспечивается поддержкой анализа приоритетов. Например, спорадические сообщения с короткими конечными сроками, но длинным временами между прибытиями могут легко описаны в этой системе. Также возможно включать многие другие модели ошибок и предсказывать время ожидания сообщения для этих моделей.

7. Литература

[1] "Electronic Exit from Spaghetti Junction", Independent on Sunday (Business), p.5 (April 3rd 1994)

[2] ISO/DIS 11898, "Road Vehicles - Interchange of Digital Information - Controller Area Network (CAN) for High Speed Communication", (February 1992).

[3] Kopetz, H., "A Solution to an Automotive Control System Benchmark", Institut fьr Technische Informatik, Technische Universitдt Wien, research report 4/1994 (April 1994)

[4] "Class C Application Requirement Considerations", SAE Technical Report J2056/1 (June 1993)

[5] "Survey of Known Protocols", SAE Technical Report J2056/2 (June 1993)

[6] Tindell, K., Burns, A., "Guaranteeing Message Latencies on Controller Area Network", to appear in the proceedings of the First International CAN Conference, Germany (September 1994)

[7] Tindell, K., "Fixed Priority Scheduling of Hard Real-Time Systems", YCST 94/03, DPhil Thesis, Department of Computer Science, University of York (1993).

[8] Audsley, N., Burns, A., Richardson, M., Tindell, K. and Wellings, A., "Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling", Software Engineering Journal 8(5) pp. 284- 292 (September 1993)

[9] Burns, A., Nicholson, M., Tindell, K. and Zhang, , "Allocating and Scheduling Hard Real-Time Tasks on a Point-to-Point Distributed System", Proceedings of the The Workshop on Parallel and Distributed Real-Time Systems, pp. 11-20, Newport Beach, California (April 13-15 1993).

[10] Burns, A., Tindell, K., Wellings, A., "Fixed Priority Scheduling with Deadlines Prior to Completion", Proceedings Sixth Euromicro Workshop on Real-time Systems, IEEE Computer Society Press (June 1994)

[11] Leung, J. Y. T., and Whitehead, J., "On The Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks", Performance Evaluation 2(4), pp.237-250 (December 1982)

[12] Lehoczky, J., Sha, L. and Ding, Y., "The Rate Monotonic Scheduling Algorithm: Exact Characterisation and Average Case Behaviour", Proceedings of the Real-Time Systems Symposium (December 1989).

[13] Tindell, K., Burns, A., Wellings, A., "Analysis of Hard Real-Time Communications", Real-Time Systems (to appear); also appears as technical report YCS 222, Department of Computer Science, University of York (January 1994)

[14] Tindell, K., and Clark, J., "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems", Microprocessors and Microprogramming 40, pp. 117-134 (1994)

Перевод : *Алексей Эстеркин, Роман Жуков, Михаил Кондратьев.*

ИТМО, 1997.
