# SpaceWire: взгляд со стороны
## Часть 1

**Валерий Журавлёв, Александр Немытов, Юрий Осипов, Андрей Першин** (Москва)

**В статье рассмотрена возможность применения протокола SpaceWire для построения нагруженных сетей реального времени с периодом задач 10…100 мс (100...10 Гц). Наглядно показано, как унаследованные особенности от прародителя – сетей транспьютеров: использование избыточного числа соединительных проводов, способ адресации, механизм червячной маршрутизации, слабая защищённость по отношению к аппаратным и программным сбоям, делают SpaceWire непригодным для использования в критически важных приложениях реального времени.**

Протокол SpaceWire ведёт своё начало [1] от транспьютерной технологии. Транспьютер – это компьютер на кристалле, который связывается со своими соседями – такими же транспьютерами – несколькими линками. Архитектура транспьютеров, наверное, красивейшая в истории вычислительной техники, была пронизана от аппаратуры до языка программирования Оккам, единой идеологией – теорией взаимодействующих последовательных процессов Хоара [2].

Линки, связывающие транспьютеры, представляли собой проводники менее 10 дюймов длиной. Транспьютер, с точки зрения сети, представлял собой одновременно и оконечную станцию, и коммутатор, а сама сеть была своего рода одним многопроцессорным компьютером, а не коммуникационной сетью в том смысле, в каком это принято понимать теперь.

Красота не спасла мир: транспьютерные технологии быстро вымерли, и одной из причин явилось то, что программирование в условиях неконтролируемой множественности параллельных процессов слишком часто приводило к непредвиденным deadlock'ам и livelock'ам.

Понятие livelock'а менее известно, чем понятие deadlock'а – в отличие от статического невыполнения какого-то условия для продолжения процесса в deadlock'е, livelock означает перманентное динамическое невыполнение условия, например, когда высокоприоритетные сообщения, идущие непрерывным потоком, не дают пробиться низкоприоритетному сообщению.

Однако транспьютерная система обмена сообщениями выжила, претерпела некоторое развитие (в частности, длина линков была увеличена до сотен метров – см. стандарт IEEE Std 1355-1995 [3]), приобрела некоторые черты настоящей коммуникационной сети, получила гордое название SpaceWire и продолжает безмятежное процветание в области, где процессы протекают небыстро, оконечные станции немногочисленны и слабо вычислительно связаны, а именно, на борту космических аппаратов, хотя наследственную склонность к deadlock'ам и livelock'ам не утратила.

Как будет видно из дальнейшего рассмотрения, сетью реального времени она так и не стала (вопреки, например, утверждению в [4]) и решение об использовании её там, где процессы протекают быстро, например, при управлении движением носителя вокруг центра масс, может быть принято только очень отважным человеком, поскольку предсказуемость времени доставки сообщения реализовать в SpaceWire в принципе невозможно.

Несмотря на громкие заявления об использовании везде и всюду, обилие научных публикаций и различных draft'ов, в SpaceWire строго специфицировано совсем немного:

1. «Провода», уровни и сигналы.
2. Зачаток понятия «пакет».
3. Методы адресации.
4. Передача данных, включая маршрутизацию.
5. Система единого времени.
6. Протокол прямого удалённого доступа к памяти.

Собственно говоря, из этого минимума, обрезанного когда-то бритвой Оккама, и родился миф о простоте SpaceWire. Однако для построения более-менее сложной сети его явно недостаточно, и тут начинается народное творчество, чем-то напоминающее небезызвестную басню Крылова: «Однажды лебедь, рак да щука...».

Каких только верхних уровней не предлагается: и переход на кодирование 8b/10b, и подсчёт CRC для пакета, и оптическая среда передачи, и временные расписания вроде TTP (Time-Triggered Protocol), и виртуальные каналы с маркерным ведром, и гарантированная доставка, и ведение статистики с протоколированием и т.д. Надёрганные к месту и не к месту механизмы из различных стройных стандартов, причудливо смешанные между собой, вызывают ощущение лёгкого дежавю.

Очевидно, что реализация любого полномасштабного протокола потребует вполне заметных ресурсов кремния, сравнимых с другими интерфейсами, но обо всём этом почему-то стыдливо умалчивают, когда говорят об экономичности и простоте протокола SpaceWire.

И главное, *все эти «нововведения» не в состоянии парировать пороки базового протокола.*

В дальнейшем мы ограничимся рассмотрением только официально принятых стандартов:

- ECSS-E-ST-50-12C – «SpaceWire – линки, узлы, коммутаторы и сети» [5];
- ECSS-E-ST-50-52C – «SpaceWire – протокол дистанционного доступа к памяти» [6] (RMAP);
- ECSS-E-ST-50-53C – «SpaceWire – протокол передачи CCSDS-пакетов» [7] (CCSDS – Consultative Committee for Space Data Systems – Консультативный комитет по космическим системам передачи данных).

### Провода

Если главному конструктору космического аппарата предложить взять на борт пару килограммов балласта – он посмотрит на вас как на «своеобразную» личность. Но если эта пара килограммов именуется «международным стандартом», то вопрос

«зачем?» даже не возникнет. Такой вот парадокс.

Во времена, когда вопрос о реализуемости проекта решался использованием в нём 100 или 200 вентилей, физический уровень SpaceWire был, своего рода, шедевром, позволяя в несколько раз экономить кремний и обеспечивать невиданные скорости передачи. Увы, расценки с тех пор несколько поменялись, и 100 лишних вентилей не заметит никто, впрочем, как и 100 000. А 4-проводная линия так и осталась вместо 2-проводной. Зачем?

Выдвигаемый тезис, что это сложно, потребует аналоговых блоков PLL (Phase-Lock Loop, фазовая автоподстройка частоты) в цифровом дизайне не выдерживает никакой критики. Во-первых, найти современную большую цифровую микросхему без последовательных каналов (RS-485, USB, Serial ATA, Ethernet, PCI Express, RapidIO и т.п.) практически невозможно. Научились их делать за последние 20 лет. Во-вторых, существует полностью цифровая техника oversampling, позволяющая на технологии сегодняшнего уровня без проблем реализовать реальные частоты SpaceWire.

И даже 2 провода – всегда ли нужны...

## Зачаток понятия «пакет»

Процессы в транспьютерах обменивались сообщениями по принципу «точка-точка». В этих условиях объяснимо стремление максимально упростить структуру сообщения:

- адрес;
- полезная нагрузка;
- признак конца сообщения.

Но даже в последней из моделей транспьютеров, в T9000 [8], сообщения разбивались на пакеты так, что пакеты из разных сообщений могли чередоваться при прохождении через физические линки. При переходе к SpaceWire было принято фантастическое решение: отождествление понятий «сообщение» и «пакет». Отказ от разбиения сообщения на пакеты, при *неограниченной* длине сообщения и червячной маршрутизации автоматом приводит к длительной *монополизации* канала! Единственное, чем можно объяснить такое решение, это стремлением до предела упростить протокол, но какова цена этому решению, мы увидим дальше.

## Чётность

Поскольку изначально транспьютерная сеть не предназначалась для выхода за пределы машины, линки-соединители были короткими, а, следовательно, сбои весьма редкими. С учётом дефицита кремния, защита битом чётности каждого байта представлялась достаточной. Однако, с введением понятия «пакет» и его неизбежного (несмотря на то, что в стандарте этого нет) закрытия CRC, использование чётности выглядит архаизмом, примерно на 10% снижающим эффективную полосу канала. Зачем?

## Адресация

Интересно, что на базовом уровне протокола SpaceWire включить CRC в структуру сообщения не удаётся из-за неопределённости длины адреса, обусловленной принятой процедурой обработки последнего.

Отсутствие контрольных сумм сообщения позволяло в байториентированной архитектуре транспьютеров использовать адресацию пути, наращивая, как это было принято, например, в структуре команд транспьютеров, байтовые префиксы. Содержимым байтового префикса является номер выходного порта, в который следует маршрутизовать сообщение в проходимом коммутаторе. Эти байтовые префиксы было необходимо отсекать по мере продвижения сообщения по сети на каждом коммутаторе. Таким образом, длина адресного заголовка сокращалась по мере прохождения маршрута.

В результате, когда при переходе от транспьютерной сети к локальной сети космического аппарата пришлось прибегнуть к защите пакета контрольной суммой (как, например, в протоколе RMAP [6]), вынужденно ограничились защитой только не меняющейся полезной нагрузки, оставив байты заголовка изменяющейся длины только со слабой защитой по чётности.

К чему ведут ошибки в адресе? Ошибки в заголовке приводят к тому, что сообщения попадают: либо в другие узлы, либо в несуществующий порт коммутатора. В качестве бесплатного бонуса могут стать ещё и широковещательными.

Второй случай сравнительно безобиден, сообщение просто тихо умрёт, не причинив никому вреда, а вот в первом случае, если пришедшее в узел чужое сообщение превышает по длине размер буфера для ожидаемого сообщения, то оно будет блокировано с соответствующими последствиями (к блокировкам мы ещё вернёмся), а если нет, то может занять место в буфере, так что пришедшему вслед штатному сообщению уже не хватит места и будет заблокировано уже оно.

Но допустим, что ничего подобного не случилось и чужое сообщение оказалось принятым. Как противодействовать пагубным последствиям такого приёма?

В протоколе RMAP [6], например, используется заголовок более высокого уровня, фактически дублирующий адресную информацию (правда, всего лишь 8-битный логический адрес из [7], что ограничивает число узлов для работы с RMAP до 256) для сообщения, потому что исходную адресную информацию сообщение растеряло по дороге. Такое решение выглядит, как заплатка.

## Передача данных: червячная маршрутизация – источник блокировок

Существовавшая в транспьютерах «червячная маршрутизация» неизбежно переехала в SpaceWire. Поясним, при червячной маршрутизации коммутатор сразу при опознании адресующего заголовка перекоммутирует сообщение на выходной порт, пока на входной порт продолжают поступать остающиеся байты сообщения. Таким образом, длинное сообщение может растягиваться на многие коммутаторы, чем-то напоминая червяка, что и оправдывает название этого принципа маршрутизации.

Поначалу червячная маршрутизация казалась очень удачной мыслью, поскольку, избегая буферизации в коммутаторе, с одной стороны, экономила память в кристалле коммутатора, что по временам конца 1980-х и начала 1990-х было существенно, а с другой стороны, локально сокращала время на коммутацию, пока не пришло осознание, что длинное сообщение, занимая многие порты, приводит к блокировке на некоторое время других сообщений, пытающихся получить доступ к занимаемым портам, причём этот процесс может развиваться лавинообразно. Эта ситуация иллюстрируется на рисунке 1.

## Состояние блокировки и выход из неё

Блокировка рассматривается как штатная ситуация, и вот что говорит по этому поводу один из ведущих специалистов по SpaceWire в [1, стр. 83]:

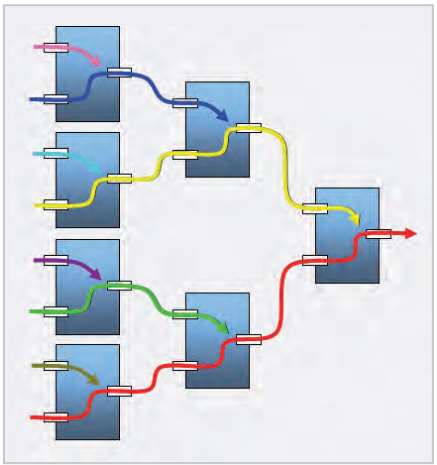

**Рис. 1. Лавинообразное каскадирование блокировок**

«Блокировка может возникнуть по нескольким причинам:

- послан большой пакет;
- получатель пакета не готов к его приёму;
- произошла какая-то авария в сети, например, выход из строя линка, так что пакет не может двигаться через этот линк.

Есть несколько стратегий, помогающих избежать блокирования в сети:

- разбивать большие пакеты на много мелких, например, посылать изображения [не целиком], а как последовательность строк изображения;
- убедиться в готовности получателя прежде, чем отсылать пакет, что может быть осуществлено реализацией механизма управления потоком из конца в конец;
- если получатель не готов принять пакет, то он может просто аннулировать его, что можно скомбинировать с механизмом перепосылки для реализации управления потоком, хотя это и может привести к неэффективному использованию пропускной способности сети, при условии, что получатель часто оказывается в состоянии неготовности;
- определить с помощью сторожевого таймера, что пакет блокирован на более длительное время, чем ожидалось, что указывает на возникновение какой-то аварии, и аннулировать блокированный пакет».

Стандарт SpaceWire [5, разд. 11.5.3, 11.5.4] предусматривает фиксацию состояния блокировки по таймауту на передачу сообщения только на исходных и приёмных узлах, что очевидно, недостаточно. Почему-то при этом для выхода из состояния блокировки даётся рекомендация организовать разрыв соединения по линку и реинициализировать его, хотя это разрушает встречный поток. Следует ли это воспринимать просто как недоработку стандарта?

Одна из ведущих организаций, поддерживающих протокол SpaceWire, Университет Данди (Шотландия), для коммутаторов предлагает следующий выход из положения [9]: на каждом порту вводится сторожевой таймер, взводимый после каждой успешной передачи байта из выходного порта, со временем срабатывания от 0,1 мс до 1 с, который по истечении тайм-аута сбрасывает остаток пакета и отправляет в канал признак конца ошибочного пакета EEP. Мы наблюдаем здесь некоторое идеологическое несоответствие стандарту, который предлагает тайм-аут на конечных узлах по передаче всего сообщения, в то время как коммутатор [9] начинает отсчёт тайм-аута заново всякий раз после передачи байта.

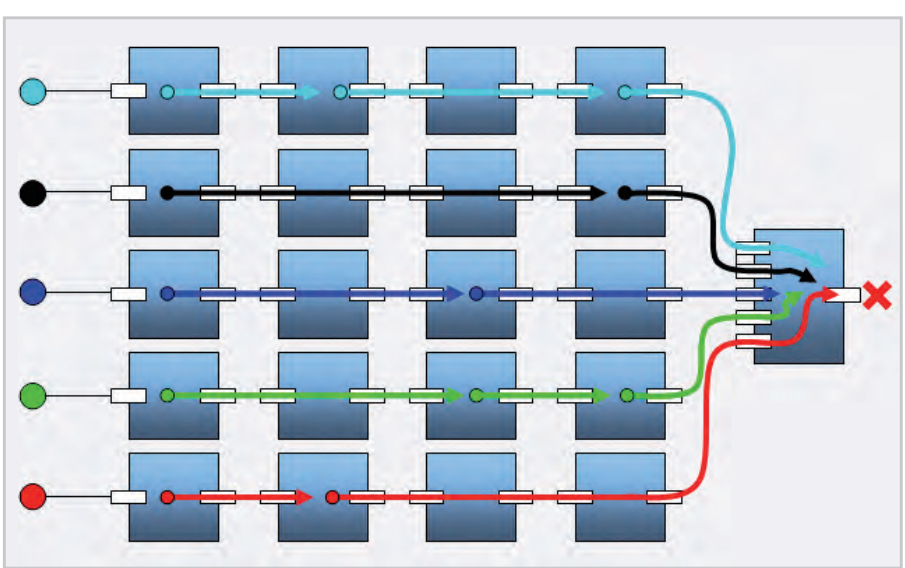

**Рис. 2. Исходные узлы, неспособные диагностировать блокировку**

Хотелось бы обратить внимание читателя на причину того, что рекомендуемые значения тайм-аутов столь велики (от 0,1 мс до 1 с) – дело в том, что при блокировке приходится ждать завершения передачи всех мешающих сообщений целиком по всему маршруту, да и не только по маршруту. Как это происходит, наглядно показывает рисунок 2.

При срабатывании сторожевого таймера в приведённом выше примере (см. рис. 1) это может привести к аннулированию множества пакетов, у которых не было бы никаких проблем с доставкой, не встреться им на пути пакет, у которого оконечная станция (узел) не готова его принимать. На птичьем языке теории сетей это называется *отсутствием логической изоляции каналов передачи.*

Даже недостижение тайм-аута не способно гарантировать отсутствие проблем: предположим, узел-получатель потребляет сообщение со скоростью 8 байтов в 0,09 мс – т.е. тайм-аут ни на каком узле не достигается ($\Delta t_{отправки} < 0{,}1$ мс). Сообщение размером в 1 кбайт будет передаваться в течение 11,52 мс, иными словами, в 112,5 раз медленнее, чем на полной скорости (при пропускной способности линков 100 Мбит/с и кодировке в канале 10 бит на байт)! Среди прочего это означает, что все выходные порты по маршруту блокируются для других сообщений в сотню раз дольше.

Вот ещё любопытный сценарий: допустим, выходит из строя некоторый узел-приёмник, тогда сообщение, адресованное ему, застревает на самом близком к нему коммутаторе. Мы можем допустить, что сообщение настолько коротко, что оно уже покинуло узел-источник и располагается червяком только в коммутаторах на своём пути. Источник никаким образом не узнает об аварии, поскольку он мог бы сделать это, согласно стандарту, только если бы зафиксировал тайм-аут у себя. Но он не может этого сделать, потому что сообщение с его точки зрения уже отправлено! Поэтому в следующий раз, когда ему понадобится отправить подобное сообщение, он это сделает ничтоже сумняшеся! И снова червяк сообщения будет блокировать на время тайм-аута занятые порты. Узел-источник сможет узнать об аварии, только если весь маршрут будет заполнен и у него начнут скапливаться неотправленные сообщения.

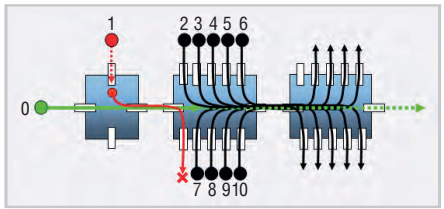


**Рис. 3. Пример динамического запирания**

**Параметры передаваемых сообщений**

| Номер узла | Период | Размер сообщения | Минимальное время прохождения через линк | Приоритет | Цвет на рисунке |
|---|---|---|---|---|---|
| 0 | 1 с | 10 Кбайт | 1 мс | Низкий | Зелёный |
| 1 | 10 мс | 16 байт | 0,0015 мс | Высокий | Красный |
| 2–10 | 1 с | 10 Кбайт | 1 мс | Низкий | Чёрный |

А теперь вообразим, что, скажем, 5 узлов будут слать свои сообщения в неисправный узел-приёмник. Можно представить себе, как будет лихорадить сеть (см. рис. 2)! Таким образом, в зависимости от применяемых методов борьбы с блокировками (или их отсутствия) мы сталкиваемся с самыми разнообразными пагубными последствиями: от частично зависшей или чрезвычайно медленно работающей системы до потери групп сообщений.

Далее рассматривается несколько примеров с разными механизмами возникновения блокировок.

### Пример блокировки: перманентное динамическое запирание

Рассмотрим сеть, состоящую из 3 коммутаторов и 24 узлов (см. рис. 3) со скоростью передачи в линии 100 Мбит/с. Предположим, что в коммутаторах реализована схема арбитража по приоритетам.

Предположим также, что каждый из передатчиков периодически с постоянным периодом отсылает сообщения постоянной длины, причём параметры отсылаемых сообщений будут соответствовать заданным в таблице.

Будем предполагать, что все тайм-ауты в системе одинаковы и определим, прежде всего, величину минимального тайм-аута на самом левом коммутаторе. Для этого рассмотрим сценарий, изображённый на рисунке 4.

Сначала генерируются девять сообщений 2–10 (на рисунке 3 они изображены чёрным цветом) и начинается их поочерёдная передача в крайний правый коммутатор, вслед за этим через очень небольшое время генерируется сообщение 0 (зелёный цвет) и оказывается блокированным на 9 мс в среднем коммутаторе. Наконец, сразу после генерации сообщения 0 генерируется сообщение 1 (красный цвет) и ждёт 10 мс, пока не пройдёт блокирующее его низкоприоритетное сообщение 0, а это произойдёт, когда пройдут опережающие 9 чёрных сообщений (9 мс) и само зелёное (1 мс). Таким образом, минимальный тайм-аут на левом коммутаторе должен составлять 10 мс, чтобы сеть нормально функционировала при исправном узле-приёмнике сообщения 1.

Обратите внимание на парадоксальный эффект, создаваемый в этом случае червячной маршрутизацией: высокоприоритетное сообщение в соседний коммутатор пройдёт только тогда, когда пройдут все низкоприоритетные сообщения во всей системе!

Предположим теперь, что узел-приёмник красного сообщения 1 (отмечен красным крестиком на рисунке 3) завис. Тогда красное сообщение застрянет в среднем коммутаторе (а, значит, и в левом) на время тайм-аута, т.е. на 10 мс и запрёт зелёное сообщение на то же время. Но к тому моменту, как для заблокированного красного сообщения истечёт тайм-аут, и оно будет аннулировано, будет послано ещё одно красное сообщение, а поскольку оно более приоритетно, нежели зелёное, то оно вновь запрёт застоявшееся зелёное сообщение. Рано или поздно зелёное сообщение будет аннулировано. И так вновь и вновь будут аннулироваться красные сообщения по уважительной причине – из-за неисправности его приёмника, а зелёные сообщения – из-за livelock'а (динамической блокировки). Ситуацию иллюстрирует рисунок 5. Обратите внимание на то, что поведение сообщений 1–10 и взаимные фазы сообщений 0 (зелёные) и 1 (красные) уже никакой роли не играют.

Резюмируем: при предполагаемой занятости сети, не превышающей 1%, возможна ситуация полного блокирования передачи из одного узла в другой несмотря на тот факт, что компоненты аппаратуры по трассе передаче исправны.

### Пример блокировки: систематическая

Рассмотрим несколько иную схему, приводящую к livelock'у (см. рис. 6). Редко происходящие конфликты могут оказаться синхронизованными так, что некоторые сообщения не имеют шанса быть доставленными, несмотря на то, что аппаратура на всех участках его маршрута исправна, а сеть практически не загружена.

В схеме 3 коммутатора и 3 оконечных станции-передатчика (узла). Будем считать, что установленная величина тайм-аута на коммутаторах всей системы

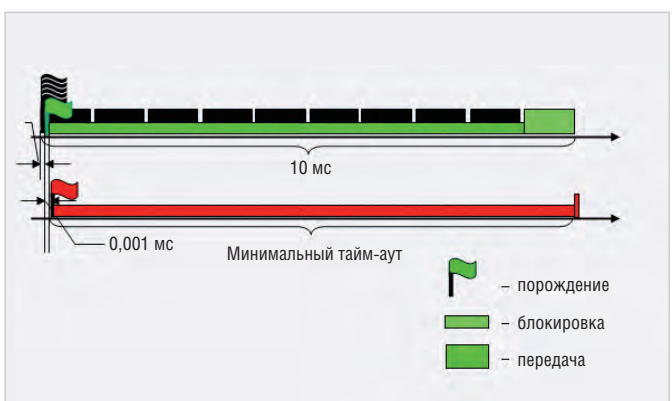


**Рис. 4. Вычисление минимального тайм-аута для схемы на рисунке 3**

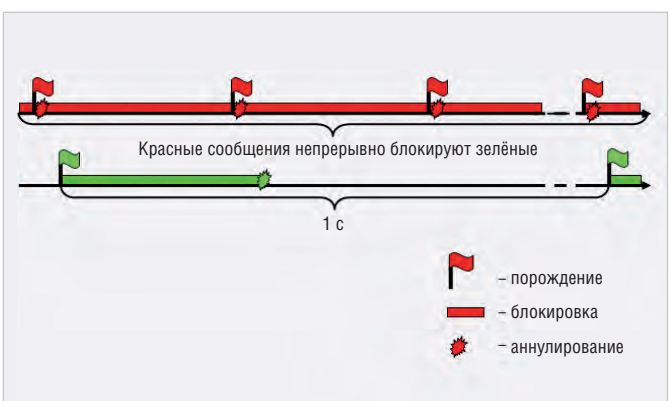


**Рис. 5. Временна́я диаграмма для примера динамического запирания**

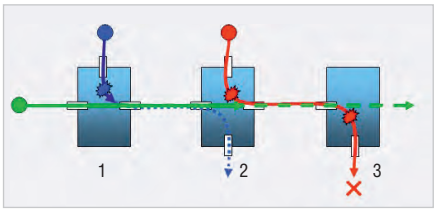


**Рис. 6. Схема синхронного столкновения**

одинакова. Все 3 оконечных станции с одинаковым *очень* большим периодом посылают одноприоритетные сообщения. Красный узел посылает сообщение среднего размера в зависший приёмник. Зелёный узел посылает достаточно длинное сообщение (но такое, чтобы время его передачи по линку в ситуации, когда отсутствуют какие бы то ни было помехи, было меньше величины тайм-аута, и, таким образом, позволяло в этой ситуации синему сообщению благополучно дождаться разблокировки). Размер синего сообщения роли не играет. Фазы отсылки сообщений могут оказаться такими, как на рисунке 7, в результате чего получим следующий эффект: красное сообщение застревает в коммутаторе 3 из-за неисправности приёмника, в него через некоторое время, но так, чтобы тайм-аут у красного сообщения ещё не истёк, в коммутаторе 2 утыкается зелёное сообщение и почти сразу в коммутаторе 1 синее сообщение блокируется зелёным.

По истечении таймаута красное сообщение аннулируется, а зелёное, хотя и подзастряло, но из-за того, что оно приостановилось несколько позже красного, аннулировано по таймауту не будет и возобновит своё движение вправо. Синее же сообщение, которое будет ожидать суммарно в течение части тайм-аута, обусловленного приостановкой зелёного сообщения, и в течение времени досылки длинного зелёного сообщения, в конечном счёте тоже будет аннулировано. На следующем цикле ситуация в точности повторится. И так снова и снова – ни одно синее сообщение никогда доставлено не будет. Ситуацию иллюстрирует рисунок 7. Он несколько упрощён, но даёт общее представление о происходящем.

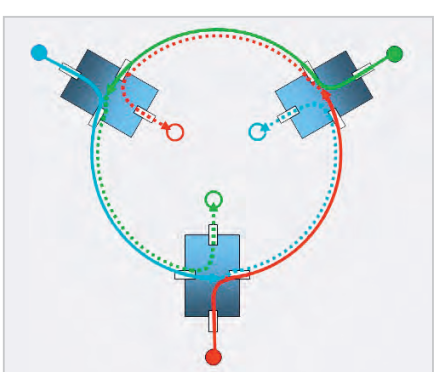

**Рис. 8. Deadlock на циклической сети**

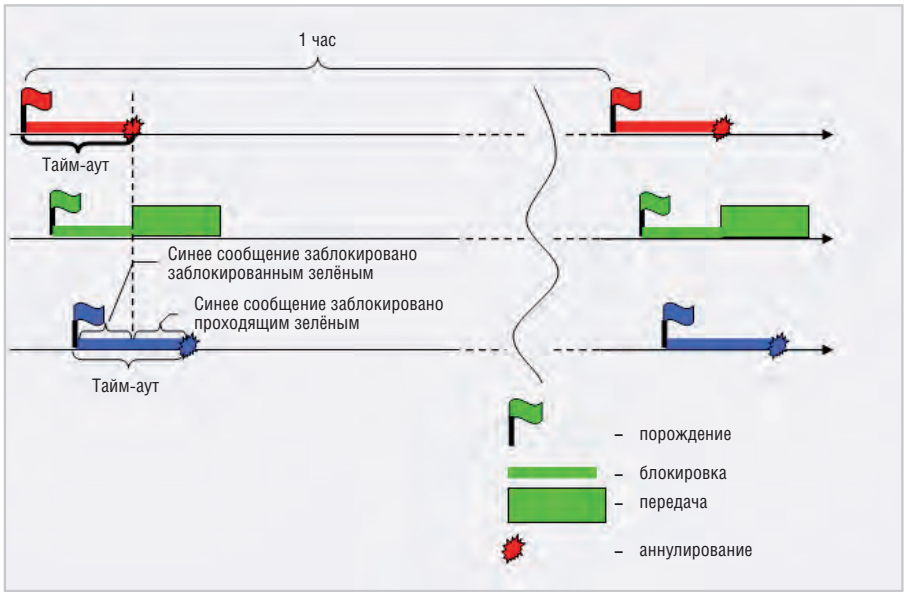


**Рис. 7. Временна́я диаграмма синхронизированных столкновений**

Резюмируем: в сколь угодно мало нагруженной сети, в случае выхода из строя одного узла, передача некоторых сообщений может быть полностью заблокирована.

### Циклы в сети – источники deadlock'ов

В стандарте очень много рассуждений про сети, содержащих циклы, даже введён ряд механизмов по обеспечению их работы. Однако забыта одна «маленькая» деталь: склонность сетей с циклами к deadlock'ам.

Рассмотрим классический пример, где deadlock появляется без каких-либо зависших приёмников и сбоев в сети. Все три сообщения стартуют одновременно с постоянным периодом. На рисунке 8 узлы-передатчики изображены заполненными кружками, а соответствующие узлы-приёмники (увы, недостижимые) пустыми кружками того же цвета. Ни одно сообщение в такой системе доставлено не будет.

Таким образом, все введённые механизмы оказываются бессмысленными, ибо сети, содержащие циклы, просто *нельзя* использовать в рамках данного стандарта.

## Заключение

Мы привели лишь несколько очевидных примеров возникновения блокировок. В реально сложной сети причины зависаний могут оказаться куда разнообразнее, и большой вопрос: когда они будут найдены – на стенде или спустя некоторое время после старта носителя.

Во второй части статьи будут рассмотрены вопросы передачи данных в стандарте SpaceWire, а также механизм синхронизации времени.

## Литература

1. *Steve Parkes.* «SpaceWire User's Guide». STAR-Dundee Limited, 2012.
2. *Хоар Ч.* Взаимодействующие последовательные процессы. – М.: Мир, 1989.
3. IEEE Std 1355-1995 Standard for Heterogeneous InterConnect (HIC) (Low Cost Low Latency Scalable Serial Interconnect).
4. *Шейнин Ю., Солохина Т., Петричкович Я.* Технология SpaceWire для параллельных систем и бортовых распределенных комплексов // Электроника. Наука. Технология. Бизнес. 2006. Вып. 5.
5. ECSS-E-ST-50-12C, «SpaceWire – Links, nodes, routers and networks», ECSS Secretariat ESA-ESTEC, Requirements & Standards Division, Noordwijk, The Netherlands, 31 July 2008.
6. ECSS-E-ST-50-52C «SpaceWire – Remote memory access protocol» 5th February 2010.
7. ECSS-E-ST-50-53C «SpaceWire – CCSDS packet transfer protocol» 5th February 2010.
8. «The T9000 Transputer Hardware Reference Manual», INMOS, SGS-Thomson Microelectronics Group.
9. «SpW-10X SpaceWire Router. User Manual», University of Dundee/Austrian Aerospace/ESTEC, January 7, 2015.

# GIGAVAC®

передовые технологии коммутации


- Реле и контакторы полностью герметичны по стандарту IP67 и IP69
- Компактный размер и различные варианты монтажа обеспечивают высокую гибкость при проектировании
- Рабочее напряжение реле до 100 000 В
  Рабочий ток реле до 150 А
- Рабочее напряжение контакторов до 1 500 В
  Рабочий ток контакторов до 1 500 А
- Возможна модификация реле под требования заказчика

☎ +7 (495) 925-50-15
✉ info@et-complex.ru

Официальным представителем и дистрибьютором продукции Gigavac в России и СНГ является фирма ООО "ЭТ-Комплекс"

et-complex.ru
gigavac.com