2.5. Настройки теста

Автор: Сергей
06.05.2022 17:55 - Обновлено 25.04.2023 12:54

Ранее...

Используя логичный, прямо по Станиславскому, подход о сплошном стремлении к сверхзадаче, мы решили основную часть проблем включения дополнительных агентов или использования встроенных в ПО, создав правильную схему . Настало время поговорить о настройках.

Начнём с обсуждения, каким протоколом следует пользоваться для расчёта *метрик* . Для этого вспомним нашу сверхзадачу — дать экономически обоснованный путь для отслеживания качества на сети . Она может быть выполнена только при использовании широко распространённых протоколов, то есть TCP, UDP, ICMP. Экзотические способы можно, конечно, рассматривать, ведь тот же UDP-lite уже давно существует, вот только их широкое внедрение будет либо затруднено, либо мы попадём в ловушку одного-двух производителей, а это явно то, чего хочется любому оператору связи в последнюю очередь.

Далее докажем, что для создания *теста* наиболее рационально использовать прикладное ПО, а не системное. Многие производители настойчиво рекомендуют путь соединения с ядром ОС с разной аргументацией, но отнюдь не мы. Итак:

1. Реализация в виде модуля ОС сразу ограничивает нас только определёнными аппаратными платформами. Так как универсальная системная разработка под несколько ОС будет стоить дорого, как при заказе сторонней организации, так и внутри. Неразумно тратить бюджет, там где можно его сэкономить.

2. Мультиплексирование потоков трафика, например, для случая одновременного запуска *тестов* в одном или разных направлениях, теперь возлагается на наше ПО. Сможет ли разработчик повторять действия создателей ОС, это большой вопрос.

3. Хотя с первого взгляда кажется, что при системной разработке может быть компенсировано время переключения контекста ОС при подсчёте задержек, это не совсем так. Ход часов в ОС всё равно зависит от шага таймера, меньше этого получить метки времени нереально. Подменять же системный таймер — это большая глупость. Ну и, строго говоря, более точные метки времени можно получать и в прикладном ПО, хотя и не во всех ОС. Да и время переключения контекста существенно меньше времени передачи пакетов через сеть.

4. Нам важно получить *метрики* , максимально близкие к пользовательскому ожиданию. А ведь рядовой потребитель услуг работает в сети с прикладным ПО в большинстве случаев.

Исходя из всех перечисленных причин мы и полагаем, что использование обычного прикладного ПО (даже простого **ping**!) вместо системного – это не только разумный, но и эффективный способ для проверки качества сети TCP/IP.

Теперь обсудим протоколы. Очевидно. что TCP для процесса расчёта *метрик* подходит слабо, ведь он скрывает для пользовательского ПО потери. Собственно, для этого его и придумали. Однако нам требуется, чтобы проблемы на сети отражались на доставке пакетов. Поэтому TCP – неверный выбор.

Как насчёт ICMP? Ведь он, собственно, для этого и предназначен, как кажется? Да и какой протокол внутри **ping**? Однако, на прикладном уровне (впрочем, на системном тоже!) вновь возникает проблема мультиплексирования потоков при одновременной посылке и приёме пакетов, упомянутая выше. Это заставляет обрабатывать некоторое количество «мусорных» пакетов, которые изначально не предназначались данному сокету. Что увеличивает нагрузку на процессор без особой пользы. Так что, минус всё-таки есть.

В противовес TCP и ICMP у протокола UDP – одни достоинства. И все ошибки передаются на пользовательский уровень. И мультиплексирование работает отлично. Прямо на уровне ядра ОС. Да и большинство приложений использует UDP, что дополнительно позволит проверять доступность пропуска трафика того или иного вида по сети. Поэтому его мы и рекомендуем.

Итак, формулируем итог.

Требование 1.

Для расчёта метрик необходимо использовать прикладное ПО, которое реализует один из прот

Теперь очень важный момент. Необходимо решить, должны ли быть *тесты* постоянными или исполняться только время от времени. С первого взгляда, памятуя, что традиционно в связи используется длительный BER-тест, хочется запустить его аналог на постоянной основе, время от времени считывая текущие результаты. Но:

1. Это постоянная нагрузка на сеть, что не разумно. Ведь у нас сеть с коммутацией пакетов, а мы пытаемся превратить её в сеть с коммутацией каналов!

2. В BER-решении нет гибкости, так как полносвязное тестирование между всеми точками сети создать на практике нереально. А ведь именно к этому и подталкивает постоянность тестирования.

3. Начальная инициация соединения и его завершение для любого вида протокола сами по себе хорошо моделируют поведение пользователя в отличие от постоянного пропуска искусственного трафика. А служба эксплуатации работает именно для пользователя.

4. В случае регулярного пропуска мы теряем гибкость настроек, особенно P_s , B_s , C_s , которые придётся фиксировать на длительный срок. Это опять-таки усложняет жизнь.

5. Простой *алгоритм 1* на приёме усложнится. Нам кажется, лучше выбирать разумные и простые решения, чем пытаться объять необъятное.

Исходя из всего вышеперечисленного, вводим определение.

Определение 76.

$$T_{freq} \quad (\text{с})$$

время ожидания запуска следующего

Понятно, что на этот период времени нужно наложить ограничения. Так как слишком частый запуск осложнит жизнь, причём бесцельно. Фактически вновь приближая нас к BER-тестированию со всеми его «особенностями». В то же время слишком редкий запуск теста будет вести в перспективе к получению информации об ошибках от пользователя, а мы-то как раз хотим этого избежать. Поэтому формулируем следующее неравенство.

Требование 2.

2.5. Настройки теста

Автор: Сергей
06.05.2022 17:55 - Обновлено 25.04.2023 12:54

$$T_{max} < \frac{8 N_s (P_s + O_s)}{B_s} < T_{freq} < 3600$$