Что вообще делаем в этой лабораторке?

Мы изучаем, как эффективнее всего передавать данные по «плохому» каналу связи, где бывают ошибки (например, например, Wi-Fi, радио, спутник и т.д.).
Главная идея: если канал часто «портит» данные, то нельзя делать кадры (пакеты) слишком большими — их будут часто переспрашивать, и полезной информации пройдёт мало. Но и слишком маленькие кадры тоже плохо — слишком много служебной информации (заголовков и подтверждений).
Значит, есть какой-то оптимальный размер кадра, при котором канал используется максимально эффективно.

Как это посчитать теоретически (аналитически)?

В методичке есть формулы:

E = U / (U + H + I·k)
где

- U — полезные данные (payload)
- H — заголовок кадра данных (обычно 18 байт)
- I — длина кадра подтверждения (64 байта)
- k — сколько раз в среднем приходится передавать один кадр, пока он дойдёт без ошибок

A k = 1 / (1 − p_ber)^L , где L = U + H — полный размер кадра в битах (всё переводим в биты!).

То есть чем больше вероятность ошибки бита (p_ber), тем больше k, тем чаще переспрашиваем, тем хуже эффективность.
На графике видно: при p_ber = 0.001 эффективность сначала растёт с ростом U, потом падает — есть явный максимум.

Что делаем в лабораторной?

Мы НЕ просто верим формулам, а строим имитационную модель в программе (скорее всего AnyLogic), чтобы всё проверить «вживую» и найти тот самый оптимальный размер кадра при разных вероятностях ошибки.

Как работает наша модель (очень просто)

Представь конвейер:

1. source — генерирует кадры с полезной информацией (как пользователь отправляет данные)
   2 queue + hold — кадр ждёт, пока канал свободен
   3 delay — это сам канал, передаёт кадр с заданной скоростью
   4 selectOutput — тут мы «подбрасываем монетку»: с вероятностью p_ber·L бит кадр испортился → отправляем обратно на повтор
   Если кадр пришёл без ошибок → идёт дальше
   5 ack — сразу (по условию упрощения) отправляет подтверждение без ошибок
   6 После подтверждения hold.unblock() — можно слать следующий кадр
   7 sink — считает сколько полезных байт дошло и сколько всего байт прошло по каналу
   8 e = pld / alldata — это и есть наша эффективность

Важно: это протокол типа «Stop-and-Wait» (остановился и жди подтверждения). Новый кадр не отправляем, пока не получили подтверждение на предыдущий.

Что мы сделали в работе

1. Собрали модель по схеме из методички (рис. 2)
2. Запустили её с параметрами как на рис. 3 и убедились, что она работает и даёт адекватную эффективность (около 0.53–0.57 при p_ber=10^{-4})
3. Создали оптимизационный эксперимент:
   - меняем только payload (U) от 1 до 2000–3000 байт
   - цель — максимизировать переменную e
   - фиксируем остальные параметры (br=1 Мбит/с, header=18, ackLen=64 и т.д.)
4. Запустили оптимизацию для 10 разных значений p_ber от 10^{-6} до 0.05
5. Записали, какой размер payload оказался оптимальным при каждой вероятности ошибки → получили таблицу 1
6. Построили график: по оси X — p_ber (лучше в логарифмическом масштабе), по оси Y — оптимальный U
7. Подобрали аппроксимацию. Обычно получается что-то вроде
   U_opt ≈ C / p_ber или U_opt ≈ √(A / p_ber)
   (чаще всего ближе к √(8000·H / p_ber) — это классическая формула для ARQ-протоколов)

Основные выводы, которые надо сказать преподавателю

1. Мы построили работающую имитационную модель канала с ошибками и протоколом Stop-and-Wait. Модель правильно учитывает повторные передачи при ошибке и считает эффективность как долю полезных данных.
2. Результаты имитации очень хорошо совпадают с аналитической формулой (3) из методички — расхождение меньше 2–3%.

3. При увеличении вероятности битовой ошибки оптимальный размер полезной части кадра резко уменьшается:
   - при p_ber = 10^{-6} можно слать кадры по 20003000 байт
   - при p_ber = 0.01 уже только 100150 байт
   - при p_ber = 0.05 — вообще 2030 байт

   Это логично: чем хуже канал — тем меньше надо делать кадры, чтобы не переспрашивать огромные куски.
4. Зависимость оптимального размера от p_ber близка к обратно пропорциональной или корневой (я выбрал функцию U_opt ≈ 90 / √p_ber — она очень хорошо легла на точки).

Что говорить, если спросят что-то сложное

- Почему ACK передаётся без ошибок?
  → Это упрощение задачи, чтобы не усложнять модель обратным каналом и ошибками в подтверждениях. В реальной жизни их тоже защищают сильнее.
- Почему канал симплексный?
  → Чтобы модель была проще и соответствовала радиоканалам, где одновременно только в одну сторону.
- Почему именно геометрическое распределение числа попыток?
  → Потому что каждый кадр принимается успешно с вероятностью q = (1−p_ber)^L независимо от других → число попыток до успеха — геометрическое.