Multiplexer

Piotr Findeisen, piotr.findeisen@students.mimuw.edu.pl

Multiplexer

- co to jest
- dlaczego powstał
- założenia
- decyzje projektowe
- architektura
- garść statystyk

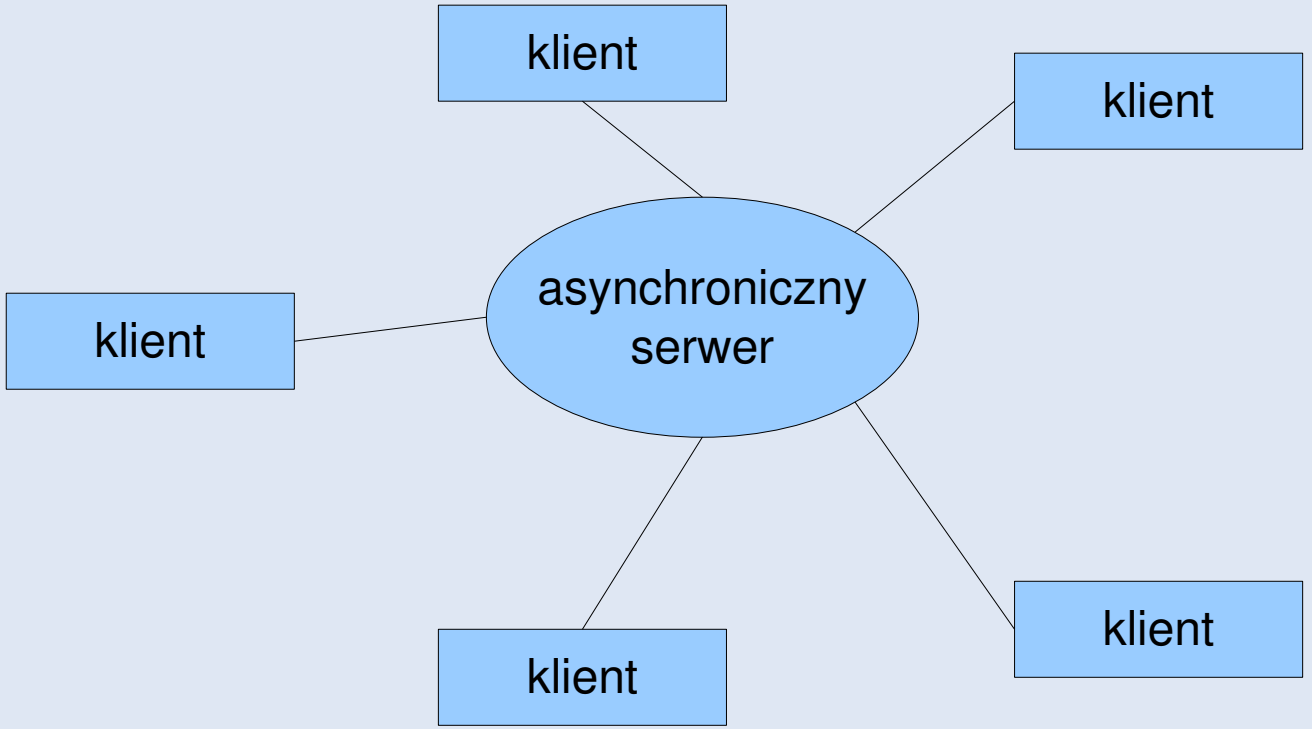
po co i dlaczego

- budujemy stronę[1] *carrier grade*
  - docelowo: brak *downtimów*
  - środowisko rozproszone
- rozwiązania typu „enterprise message bus” ;)
- Spread?
- inne?

[1] www.azouk.com (release wkrótce)

historia

- Multiplexer bazuje na doświadczeniu:
  - Pythonowego multiplexera Krzysztofa Kulewskiego
  - Bramki z Gemiusa (brałem udział w rozwoju)



pożądane cechy

- wysoka dostępność
- pewność dostarczenia
- duża przepustowość
- małe opóźnienia
- model *publish-subscribe*
- automatyczna gwarancja spójności

wilk i owca...

- **wysoka dostępność**
- pewność dostarczenia
- duża przepustowość
- małe opóźnienia
- model *publish-subscribe*
- **automatyczna gwarancja spójności**

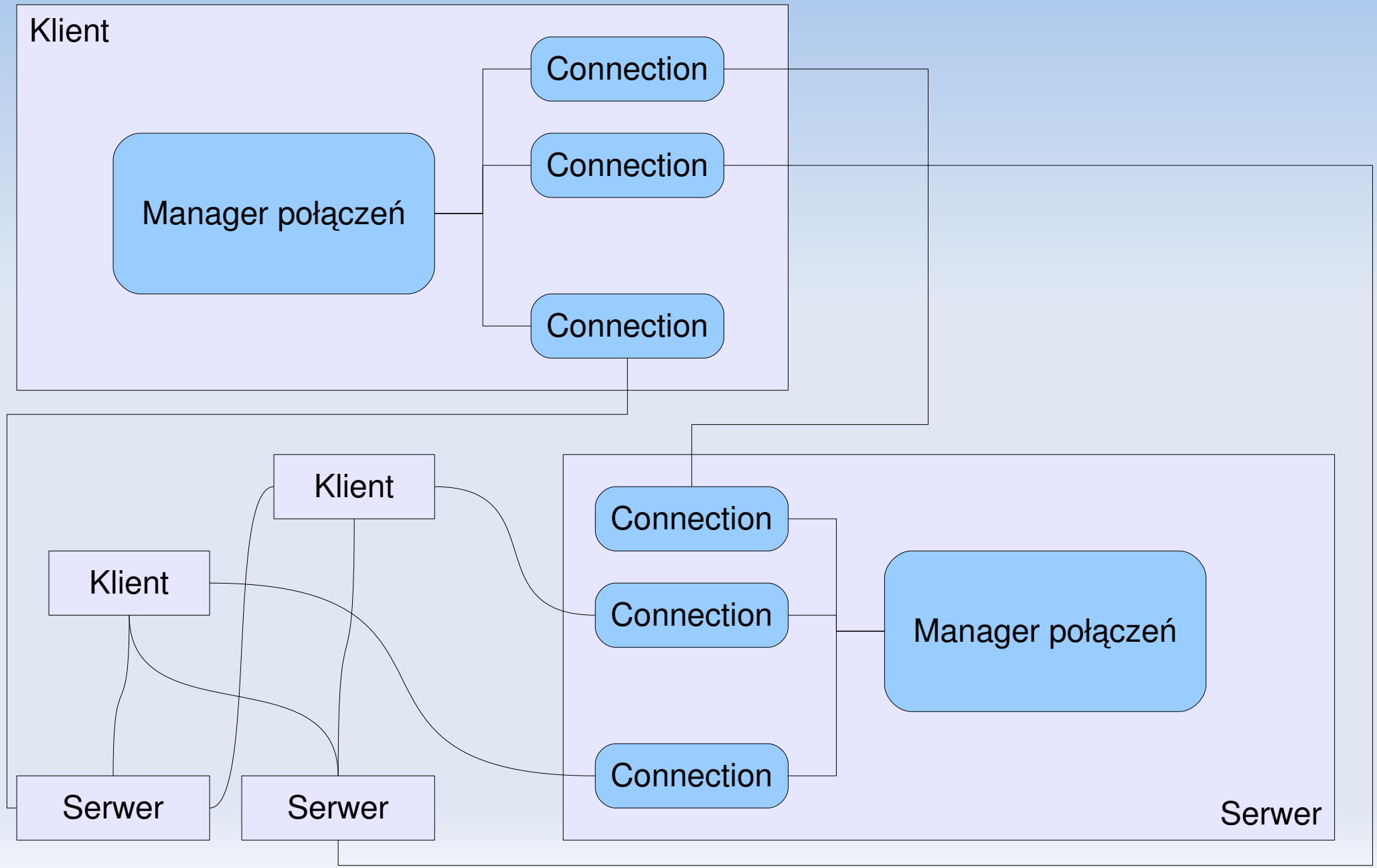
nasz wybór

1. wysoka dostępność
2. pewność wykonania zadania
   (nie: *dostarczenia wiadomości*)
3. duża przepustowość
4. małe opóźnienia
5. model *publish-subscribe*

nasz wybór cd.

- serwer w C++
- dostęp z różnych języków programowania
  - prosty, przenośny protokół komunikacyjny
  - obecnie Python i C++
- rozszerzalność
  - implementacji
  - protokołu

architektura



routing

- reguły na podstawie typu pakietu
  - do jednego/wszystkich odbiorców danego typu
  - dodatkowe meta info

```
type {
    type: 117
    name: "SEARCH_COLLECTED_LOGS_REQUEST"
    comment: "payload is SearchCollectedLogs; returns LogEntriesMessage"
    to {
        peer: "LOG_COLLECTOR"
        whom: ANY
    }
    report_delivery_error: true
}
```

routing – cd.

- reguły na podstawie typu pakietu
  - do jednego/wszystkich odbiorców danego typu
  - dodatkowe meta info
- wiadomości wprost do adresata
- reguły razem z pakietem

- usuwanie duplikatów po stronie klienta

technologie

- Google Protocol Buffers
  - rozszerzalny format wymiany danych
  - nie XML
- ASIO (Asynchronous C++ library)
  - *... library should facilitate the development of network applications that scale to thousands of concurrent connections ...*
- Boost.Python

Google Protocol Buffers

- binarny, efektywny format
- typy
  - numeryczne
  - wyliczeniowe
  - string, byte array
  - listy
  - klasy (wiadomości) złożone z typów prostych i klas
- generator klas dla C++, Pythona, Java, etc.

Protobuf: message DL

```
message MultiplexerMessage {
    // packet ID
    required uint64 id = 1;

    // ID of the sender (generally required)
    optional uint64 from = 2;
    optional uint64 to = 3;

    // message type
    required uint32 type = 4;

    // the actual message
    optional bytes message = 5;

    // packet ID to which we reply
    optional uint64 references = 7;

    // when the packet was sent
    optional uint32 timestamp = 6;
    ...
}
```

protokół Multiplexera

bajty 0-3	**bajty 4-7**	**...**
długość (little endian)	suma kontrolna (little endian)	protocol buffer (format binarny)

ASIO (Asynchronous C++ library)

- równoległość bez wątkowości
- pozwala wydajnie używać sieci
- cóż, C++ nie jest takie *agile*...

```
asio::async_read(socket_,
       asio::buffer(incoming_message_->get_header_buffer()),
       boost::bind(&Connection::_handle_read_header,
          this->shared_from_this(),
          asio::placeholders::error,
          asio::placeholders::bytes_transferred)
   );
```

Boost.Python

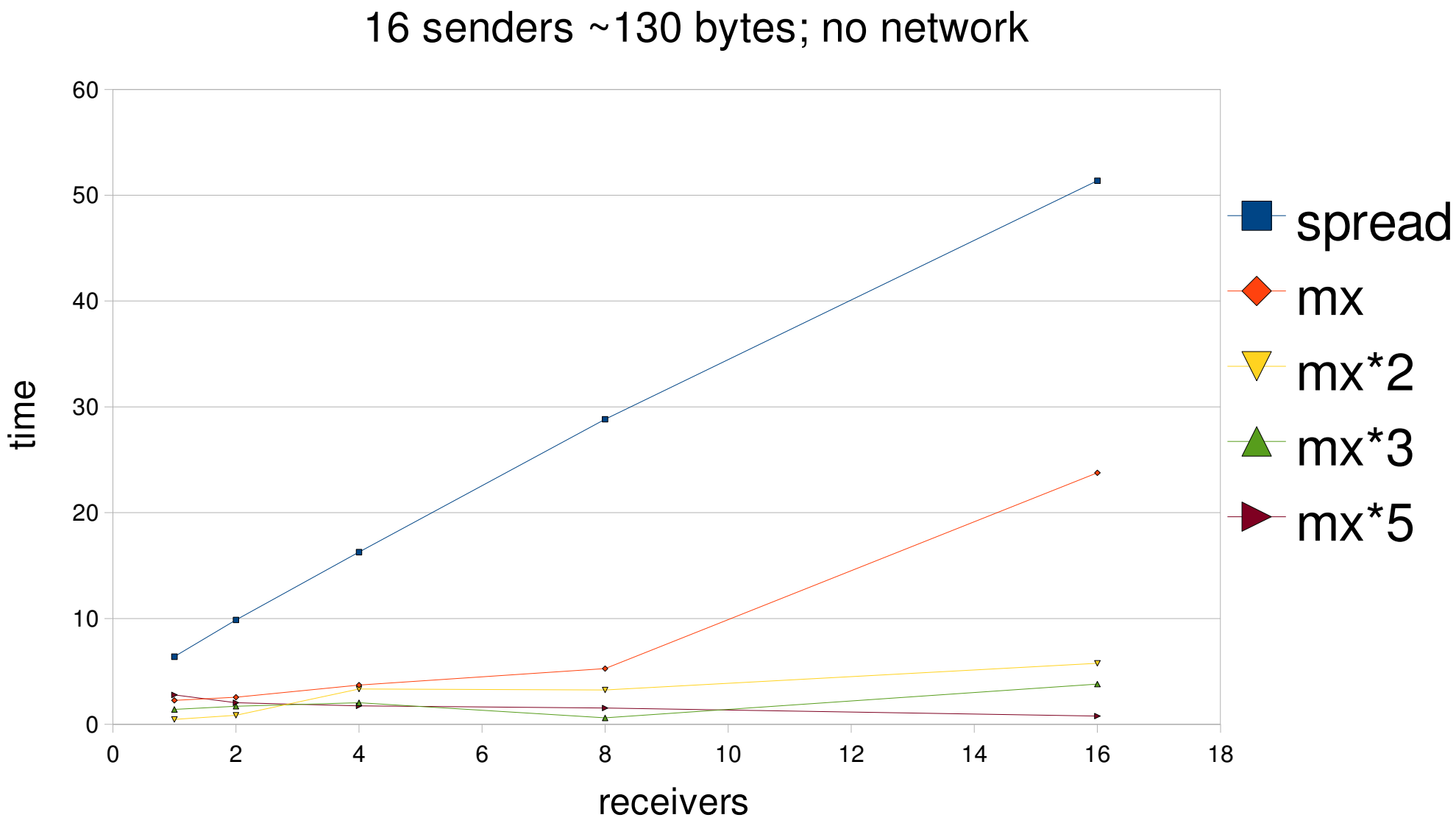
- skleja C++ i Pythona
- eksportuje obiektowe interfejsy
- translacja wyjątków
- brak dodatkowego kroku kompilacji (*pure C++*)

- → biblioteka dla Pythona jest w C++
  i używa ASIO

wydajność – przykład

- komunikacja: rozsyłanie powiadomień
- klient wysyła powiadomienia do 16 subskrybentów; 1 Multiplexer
  - bez sieci
- 10000 powiadomień, 34 bajty każda w 4.5 s
  - → ponad 37 500 wiadomości na sekundę !
  - na jednym procesorze Intel Xeon 2.66GHz

wydajność – przykład



wydajność – przykład

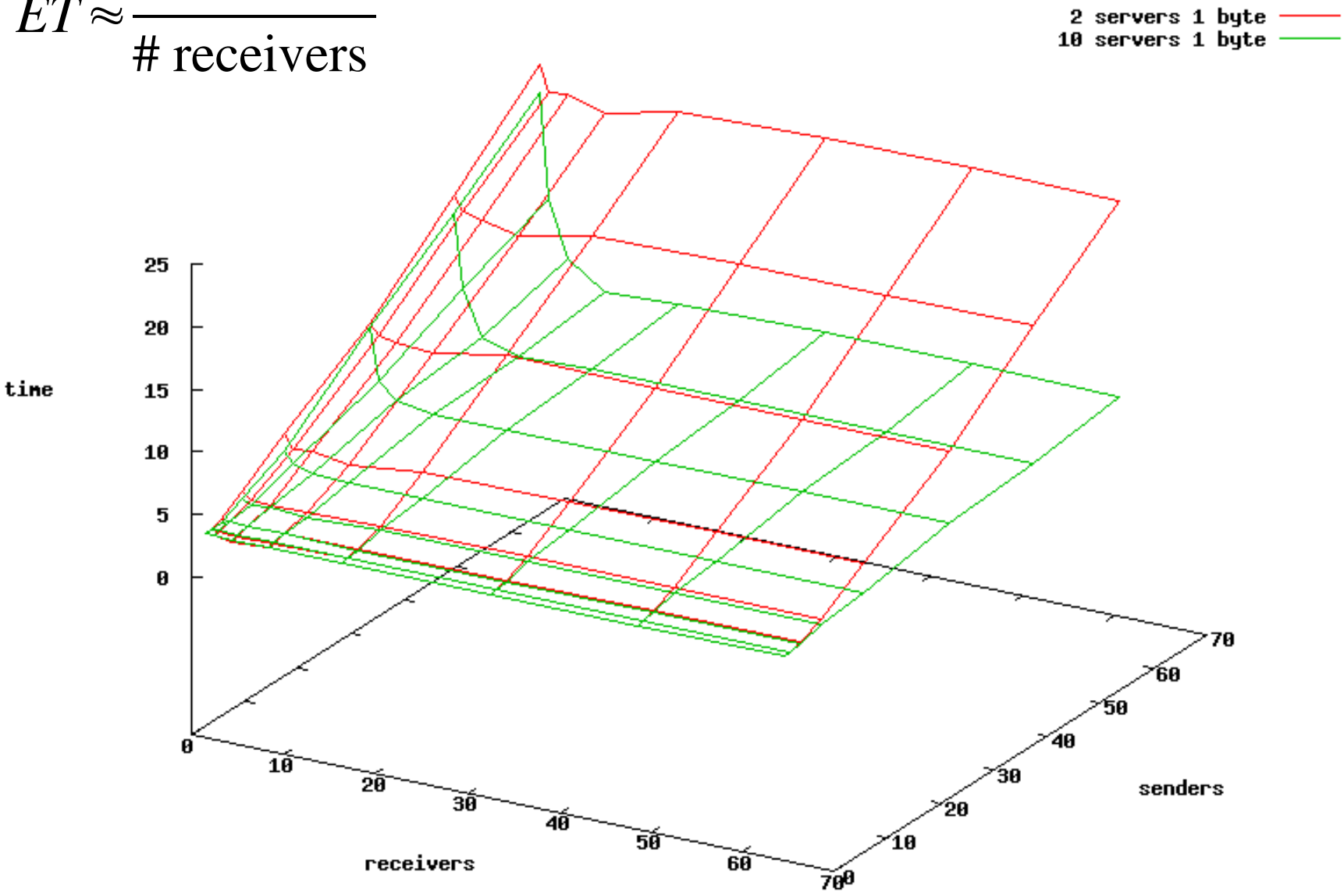
- komunikacja klient-serwer
- 1 klient; wysyła żądanie dopiero po otrzymaniu odpowiedzi na poprzednie
- 10000 zapytań, 34 bajty każde w 3.46 s
  - → 5780 wiadomości na sekundę (0.18ms)
  - end to end
  - na zwykłych maszynach z labów (Intel Core 2.40GHz)

wydajność – przykład

- komunikacja klient-serwer
- zasady jak poprzednio
- 15 Multiplexerów, 64 klientów i 64 serwery
- 64 × 10000 zapytań, 34 bajty każde w 6.37 s
  - → w sumie 200 000 wiadomości na sekundę !
  - 13 400 /s na jednym Multiplexerze

wydajność – przykład

$$ET \approx \frac{\#\ \text{senders}}{\#\ \text{receivers}}$$



lepimy bałwana?

Questions?
Answers?