LoXiM — Projekt protokołu komunikacyjnego - wersja 2.0 Klient↔Serwer

Piotr Tabor (pt214569@students.mimuw.edu.pl)

4 czerwca 2008

Wersja: 0.8

Historia

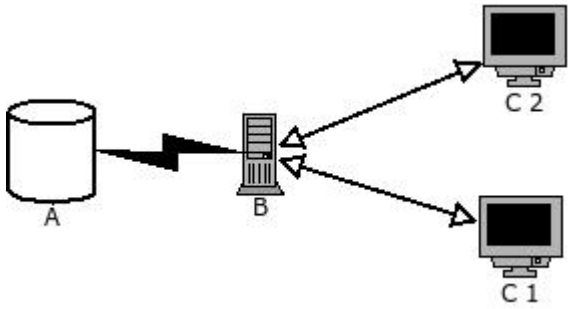
Data	Wersja	Autor	Zmiany
2006-12-30	0.1	Piotr Tabor	Pierwsza wersja dokumentu
2006-12-31	0.2	Piotr Tabor	Poprawki według uwag dra hab. Krzysztofa Stencla
2007-01-05	0.3	Piotr Tabor	Poprawki głównie literówek wykrytych podczas seminarium 2007-01-04
2007-01-12	0.4	Piotr Tabor	Poprawki z seminarium 2007-01-11. Wprowadzenie typu VOID. Połączenie informacji o dokonanych modyfikacjach z każdym przeprowadzonym zapytaniem
2007-02-10	0.5	Piotr Tabor	Poprawki związane z implementacją: Wymiana w nagłówku pakietu zmienno-długościowego pola varuint na stałe: uint32
2007-02-19	0.6	Piotr Tabor	Dodanie rysunku przejść pomiędzy stanami serwera
2007-02-19	0.7	Piotr Tabor	Usunięcie pakietu Q-C-CANCEL (na rzecz VQ-SC-CANCEL). Uporządkowanie i ujednolicenie kwestii odpowiedzi synchronicznych.
2007-05-10	0.8	Piotr Tabor	Połączenie VQ-SC-CANCEL z VQ-SC-ABORT. Uspójnienie rozmiarów kilku struktur. Dodanie diagramu warstw protokołu i diagramu przejść między stanami (patrz: 5.3).
2008-05-28	0.9	Piotr Tabor	Rozwinięcie kwestii stref czasowych (patrz: 3.6.1) i porównywania napisów. Poprawienie błędów składniowych. Zmiana nazwy typu z „bob” na „bytes”.

Spis treści

1 Wstęp

Dokument ten opisuje protokół wymiany danych w systemie semistrukturalnej, obiektowej bazy danych opartej na stosowym języku zapytań (SBQL). Protokół służy do komunikacji pomiędzy aplikacją kliencką, a odpowiednim oprogramowaniem bazy danych — służącym do wykonywania zapytań i umożliwiającym autoryzację użytkowników.

Zatem rozpatrzmy hipotetyczną sytuację w której baza danych - rozumiana jako zbiór danych i mechanizm jej odczytów znajduje się na komputerze A, na komputerze B znajduje się proces przetwarzania zapytań, a na komputerze C znajduje się aplikacja chcąca korzystać z danych poprzez zadawanie zapytań. Opisany w poniższym dokumencie protokół służy do dwukierunkowej komunikacji pomiędzy aplikacjami działającymi na komputerach B i C, i nie nadaje się do komunikacji pomiędzy komputerami A i B.



Rysunek 1: Diagram elementów systemu

Obecnie w systemie „LoXiM" część „A" (storage) i „B" (executor) są zintegrowane w jednej aplikacji.

Dokument ten został opisany na potrzeby projektu „LoXiM", ale z poszanowaniem warunków umowy licencyjnej (GPL) protokół może zostać wykorzystany w dowolnej bazie danych, której potrzeby spełnia. Rozwiązania zastosowane w tym protokole wydają się być na tyle ogólne, że z dużą pewnością mogą zaspokoić potrzeby wielu rozwiązań — także relacyjnych i nie opartych o SBQL'a.

Protokół ten stara się połączyć najlepsze cechy poniższych protokołów, jednocześnie skupiając się na wprowadzeniu możliwości pracy z danymi semistrukturalnymi:

Postgresql 10 — używanego przez PostgreSQL 8.2 [2]

TDS 8.0 — używanego przez Sybase i Microsoft SQL Server 7.5/2000 [4]

MySQL 3 — używanego przez serwer MySQL 5 [1]

1.1 Cele i założenia

Celem projektu opisanego przez ten dokument jest:

- Uzyskanie stabilnego. bezpiecznego protokołu, umożliwiającego pełne wykorzystanie obecnych możliwości systemu LoXiM.
- Uzyskanie protokołu potrafiącego pracować pomiędzy maszynami o różnych architekturach sprzętowych i programowych
- Uzyskanie efektywnego (pod względem wykorzystania sieci i CPU) i łatwo rozszerzalnego protokołu.
- Przygotowanie do implementacji sterownika JDBC w oparciu o ten protokół.

Przyjęto następujące założenia:

- Zakładamy, że komunikacja będzie się odbywała po odpornej na zakłócenia, błędy transmisji i zamianę kolejności przesyłanych danych warstwie transportowej (TCP/IP, Unix sockets, łącza nazwane Windows itp.).

  Zatem nie będziemy przeprowadzali własnej kontroli spójności przesłanych danych — pod względem sum kontrolnych itp. Jednak ze względów bezpieczeństwa kontrola logiczna będzie oczywiście przeprowadzana.

1.2 Wersjonowanie protokołu

Protokół komunikacji będzie wersjonowany przy pomocy dwóch numerów: głównego (major) i pomocniczego/pobocznego (minor). Poniższy dokument opisuje potoków w wersji 2.0, czyli numer główny 2, a pomocniczy 0. Przyjmuje się, że aplikacje posługujące się protokołem o tym samym numerze głównym są ze sobą zgodne - jedynie możliwości komunikacji są ograniczone do tych dostępnych w starszej wersji protokołu (mniejszy numer pomocniczy).

Zatem w obrębie tego samego numeru głównego można przewidzieć następujące typy zmian:

- Rozbudowanie formatu paczki poprzez dodanie na jej końcu nowych (nie obowiązkowych pól).
- Wprowadzenie nowych typów paczek — niekluczowych dla działania systemu i niemodyfikujących dotychczasowej semantyki operacji.

1.3 Licencjonowanie

2 Paczka — jednostka logiczna komunikacji

2.1 Paczka, a pakiet

Cała komunikacja będzie się opierała o przesyłanie paczek — spójnych ciągów danych o określonym formacie. Słowa „paczka" będziemy używali dla rozróżnienia i uniknięcia nieporozumień z pojęciem pakietu — który ma znaczenie na poziomie warstwy transportu (np. TCP/IP).

Zatem paczka to zbiór danych mających logiczne znaczenie w systemie LoXiM. Zupełnie nie wnikamy w to w jaki sposób paczki zostaną zorganizowane w pakiety i to zadanie pozostawiamy warstwie transportu. Jedyne co musimy zagwarantować, to fakt, że pojedyncza paczka jest spójnym obszarem danych w komunikacji.

W dokumencie tym będziemy także używali sformułowania „komunikat", które uznajemy za synonim słowa „paczka".

2.2 Budowa paczki

Każda paczka ma następujący schemat budowy:

Od - do	**Typ/Wartość**	**Zawartość**
$0 \rightarrow 0$	uint8	Stała mówiąca o typie paczki, a tym samym określająca format zawartych w nich danych
$1 \rightarrow 4$	uint32	n — stała określająca ilość danych właściwych zawartych w paczce – wyrażona w bajtach
$5 \rightarrow 4 + n$	patrz opis zależny od typu paczki	Dane właściwe paczki zgodne z formatem określonym poprzez typ paczki

Formalnie będziemy mówili, że paczka się składa z dwu-polowego nagłówka oraz ciała. Nagłówek wyznacza typ paczki i rozmiar danych właściwych w niej zawartych, a ciało — to dane interpretowane zależnie od typu paczki. Kluczową część tego dokumentu stanowi opis formatów poszczególnych paczek w zależności od ich typów.

Przyjmuje się, że rozmiar pojedynczej paczki nie może przekraczać 1MB (1'048'576 bajtów). Służy to uniknięciu sytuacji, w których odbywa się próba naruszenia bezpieczeństwa serwera poprzez przesłanie zbyt dużej paczki (doprowadzenie do wystąpienia błędu OutOfMemory), a także uniknięciu sytuacji w której traci się możliwość komunikacji asynchronicznej w skończonym czasie (np. wysłanie do serwera informacji o braku dalszego zainteresowania danymi). Wyżej wymieniona stała może być (a nawet powinna) konfigurowalna po stronie serwera.

2.3 Czemu przesyłamy długość paczki?

Przesłanie długości paczki na jej początku niesie za sobą następujące ułatwienia:

- Klient może zaalokować właściwą ilość danych w pamięci operacyjnej na przyjęcie całego komunikatu z góry — co ma pozytywny wpływ na wydajność i bezpieczeństwo (nie nadejdzie „nieskończenie" długi komunikat).

- Klient może zrezygnować z pobierana komunikatu (np. z powodu brak wystarczającej ilości pamięci, by go przyjąć) lub braku zainteresowania. Dzięki temu wie, ile bajtów musi zignorować bez ich analizy.

2.4 Mechanizmy rozszerzania protokołu

Protokół może być rozszerzany z zachowaniem zgodności wstecz poprzez dołączanie nowych pól na końcu danych paczki. Zatem strona odczytująca komunikat nie może zakładać, że po odczytaniu wszystkich pól w paczce o których wie, dotarła na koniec paczki. Możliwe, że strona ta przeczyta wszystkie pola i nie dotrze wcale do końca paczki, ponieważ w paczce występują dane dotyczące nowszej wersji protokołu. Zatem powinna ona zignorować odpowiednią ilość bajtów — wynikającą z długości danych w paczce, tak aby dotrzeć na początek następnego komunikatu.

Także możliwa jest odwrotna sytuacja. Strona wysyłająca posługuje się starszą wersją protokołu, więc wysyła komunikat nie zawierający pól wprowadzonych w nowszej wersji protokołu. Zatem strona odczytująca (która jest nowsza) powinna umieć właściwie zareagować na sytuację, gdy odczytano całą paczkę, a nie otrzymano nowych pól wprowadzonych w protokole. W tej sytuacji powinna przyjąć wartości domyślne dla tych pól.

Oczywiście odpowiednio duże zmiany i przeprojektowanie w protokole mogą wymagać stworzenia protokołu o kolejnym numerze głównym, czyli niezgodnego ze starszymi wersjami.

3 Podstawowe typy danych

3.1 Postanowienia ogólne

- Jeśli w sposób szczególny nie zaznaczono inaczej (a raczej nigdzie nie zaznaczono) wszystkie wartości zapisywane są w formacie Big-endian. W szczególności obejmuje to typy całkowitoliczbowe, rzeczywiste, oraz napisy w kodowaniu UTF-8 także stosują kolejność Big-endian.

3.2 Całkowitoliczbowe: uint8, sint8, uint16, sint16, int32, uint32, uint64, sint64

Pierwsza litera determinuje, czy mamy do czynienia z typem ze znakiem (u - unsigned), czy z typem bez znaku (s — signed). Liczba na końcu wyraża długość typu wyrażoną w bitach.

Typy są kodowane oczywiście w kolejności Big-endian.

3.2.1 varuint — Całkowitoliczbowy z kompresją (1,3,5 lub 9 bajtów)

Będzie to typ używany głównie do oznaczania długości stringów i ogólnie paczek.

Rozwiązanie techniczne zostało zaczerpnięte z protokołu serwera MySQL [1].

Idea jest taka, że krótkie stringi (< 250 znaków) będą się pojawiały najczęściej i chcemy mieć najmniejszy narzut na zapisanie długości (jedno bajtowy).

Zatem semantyka pierwszego bajtu jest następująca (w zależności od jego wartości)

0-249 Wartość ta jest jednocześnie wartością wynikową

250 Wartość jest null'em (zostawiamy dla umożliwienia stosowanie tego rozwiązania z systemami relacyjnymi)

251 Wartość ta poprzedza 2-bajtowe (uint16) pole w wartością właściwą

252 Wartość ta poprzedza 4-bajtowe (uint32) pole z wartością właściwą

253 Wartość ta poprzedza 8-bajtowe (uint64) pole z wartością właściwą. Nie należy jednak korzystać z wartości większych niż $2^{63} - 1$ (MAX_SINT64), ze względu na to, że w Javie nie są obsługiwane 8 bajtowe typy bez znaku.

3.3 string — Łańcuchy tekstu

Służy do przesyłania tekstów. Teksty te muszą być kodowane za pomocą UTF-8.

Format wartości typu string jest następujący: Najpierw idzie pole typu varuint — opisujące długość w bajtach następującego potem ciągu w UTF-8 (Big-endian).

3.4 sstring — Krótki łańcuch tekstu

Jest to tak naprawdę wariant typu string, ale ograniczony do łańcuchów nie dłuższych niż 249 bajtów. Czyli wtedy pole oznaczające długość ma jeden bajt. Jego wewnętrzna reprezentacja zupełnie się nie różni od typu string — został on wyróżniony formalnie — ze względu na uproszczenie notacji używanej przy opisach formatów poszczególnych paczek.

3.5 bytes — Dane binarne

Służy do przesyłania danych binarnych (nie koniecznie dużych — nie będziemy tego rozróżniali na poziomie protokołu).

Format wartości tego typu jest następujący: Najpierw zostaje przesłane pole typu varuint — opisujące długość w bajtach następującego potem ciągu bajtów.

3.6 Daty i czas

3.6.1 Kwestia stref czasowych

Można rozważyć dwa podejścia do przekazywania informacji o strefie czasowej:

- Przekazujemy tylko offset strefy czasowej względem GMT (czyli wartość od -14 do $+12$)

- Przekazujemy pełną informację o strefie czasowej. Wygląda na to, że nie istnieje standard ISO opisujący kodowanie dla wszystkich stref czasowych na świecie. Najlepszą bazą danych z informacjami o strefach czasowych jest baza TZ (znana też jako zoneinfo) (http://www.twinsun.com/tz/tz-link.htm). Koduje one informacja o strefie czasowej w postaci napisu: {Kontynent}/{Duże miasto} lub {Kontynent}/{Państwo}/{Duże Miszto}, np. „Africa/Porto-Novo".

  Zaletę przekazywania daty z tą pełną informacją, jest to, że dopiero na jej podstawie system informatyczny jest wstanie prawidłowo dodać pewną ilość czasu (np. 36 godzin) do danej daty w sytuacji, gdy w międzyczasie występuje zmiana czasu z letniego na zimowy lub odwrotnie.

Nie budzi wątpliwości, że dla uniknięcia problemów w aplikacjach o dużym zasięgu terytorialnym warto by było z datą zapisywać pełną informację.

Według bazy „TZ" w chwili obecnej na świecie jest używanych 398 różnych stref czasowych. Dobry standard pozwoliłby zakodować je za pomocą trzech liter, czyli — będąc rozrzutnym — 3 bajtów. Oszczędność 2 bajtów wydaje się być więc mało uzasadniona wobec ryzyka utraty poprawności numerycznej niektórych operacji, skoro stanowi ona wzrost długości całej zakodowanej daty z 9 do 11 bajtów, czyli o 22%.

Niestety za nieformalny standard w bazach danych (PostgreSQL, MySQL, Oracle, DB2) przyjęło się zapisywać jedynie informacje o przesunięciu względem GMT, co wynika ze złożenia dwóch problemów:

- Światowy standard formatu czasu ISO8601 [6] przewiduje tylko strefę czasową zapisaną w postaci przesunięcia względem GMT.
- Brak standardu (klasy ISO) kodowania stref czasowych, co pośrednio wynika z punktu powyższego.

Z tego powodu obecna wersja protokołu zapisuje strefy czas w „standardowy" sposób, czyli w postaci offsetów. Zalecamy jednak używanie pełnej informacji o strefie czasowej w następnych wersjach protokołu.

3.6.2 DATE

Sama data.

Format następujący:

Od - do	**Typ/Wartość**	**Zawartość**
$0 \rightarrow 1$	*Year* (sint16)	Rok
$2 \rightarrow 2$	*Month* (uint8)	Miesiąc (1-styczeń, 12-grudzień)
$3 \rightarrow 3$	*Day* (uint8)	Dzień

3.6.3 TIME

Sam czas.

Format następujący:

Od - do	**Typ/Wartość**	**Zawartość**
$0 \rightarrow 0$	*Hour* (uint8)	Godzina (0-23)
$1 \rightarrow 1$	*Minuts* (uint8)	Minuta (0-59)
$2 \rightarrow 2$	*Secs* (uint8)	Sekunda (0-59)
$3 \rightarrow 4$	*Milis* (sint16)	Milisekundy (0-999)

3.6.4 TIMETZ

Czas ze strefą czasową

Format następujący:

Od - do	Typ/Wartość	Zawartość
$0 \rightarrow 0$	*Hour* (uint8)	Rok
$1 \rightarrow 1$	*Minuts* (uint8)	Miesiąc (1-styczeń, 12-grudzień)
$2 \rightarrow 2$	*Secs* (uint8)	Dzień
$3 \rightarrow 4$	*Milis* (uint16)	Milisekundy
$5 \rightarrow 5$	*TZ* (sint8)	Strefa czasowa (-14 do +12)

3.6.5 DATETIME

Data i godzina bez strefy czasowej. Format zapisu to bezpośrednio po sobie występujące formaty pól typu DATE i TIME.

3.6.6 DATETIMETZ

Data i godzina ze strefą czasową. Format zapisu to bezpośrednio po sobie występujące formaty pól typu DATE i TIMETZ.

3.7 Logiczne

Pole typu bool niesie pewną wartość logiczną. W praktyce będzie reprezentowane jako liczba typu sint8 z następującymi wartościami:

0 Fałsz

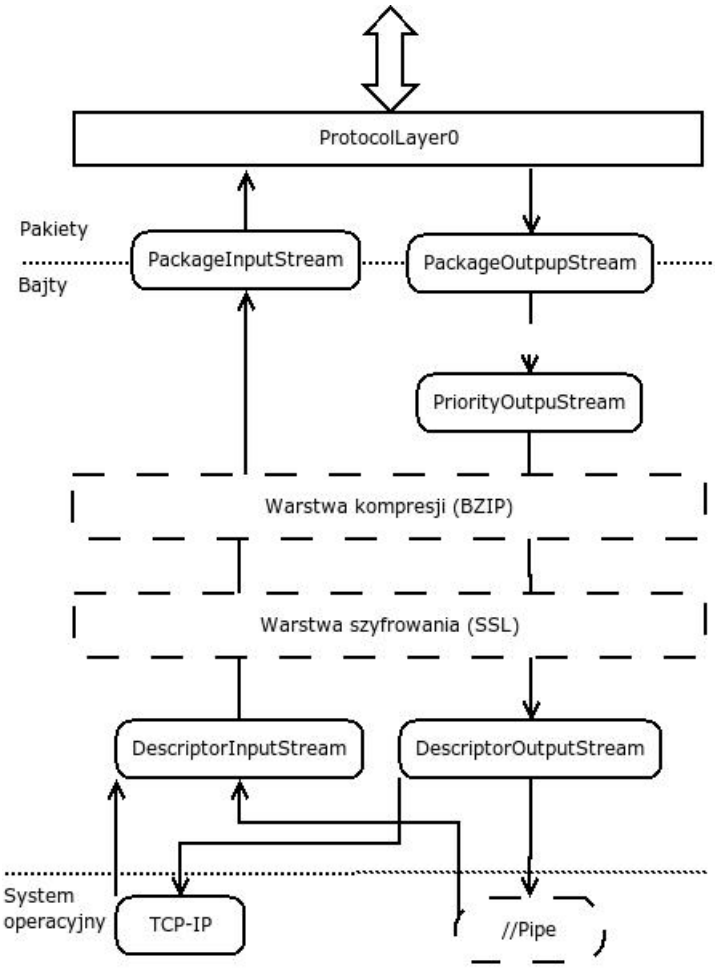
1 Prawda

3.8 Zmiennoprzecinkowe

3.8.1 DOUBLE

Podwójna precyzja, 8-bajty w kolejności Big-endian.

4 Podwarstwy



Rysunek 2: Diagram warstw

Protokół w warstwie aplikacyjnej może chodzić w kilku podwarstwach. Elementy te umożliwiają szyfrowanie i kompresję w czasie działania protokołu.

W poniższych rozważaniach rozpatrzmy sytuację najbardziej złożoną — w której zarówno szyfrowania jak i kompresja jest dostępna.

4.1 Pisanie danych

Aplikacja chce wysłać paczkę do innej aplikacji:

1. Aplikacja umieszcza (zapisuje) wysyłaną paczkę do strumienia wyjściowego.
2. Strumień ten okazuje się być strumieniem kompresującym (ZLIB), który skompresowane dane zapisuje do swojego strumienia wyjściowego.
3. Strumień ten okazuje się być strumieniem szyfrującym (SSL), który dokonuje szyfrowania danych, a następnie umieszcza je w swoim strumieniu wyjściowym
4. Strumień ten okazuje się być strumieniem odpowiedniej warstwy transportu, która to warstwa przekazuje dane przez sieć.

4.2 Czytanie danych

Aplikacja chce odczytać paczkę pochodzą od innej aplikacji.

1. Aplikacja prosi o paczkę odpowiedni strumień wejściowy.
2. Strumień ten okazuje się być strumieniem dekompresującym (ZLIB), który prosi o dane swój strumień wejściowy.
3. Strumień ten okazuje się być strumieniem deszyfrującym (SSL), który prosi o dane swój strumień wejściowy.
4. Strumień ten okazuje się być strumieniem odpowiedniej warstwy transportu i odbiera on te dane z sieci, a następnie zwraca je.
5. Strumień deszyfrujący przetwarza dane i zwraca je
6. Strumień dekompresujący przetwarza dane i zwraca je
7. Aplikacja odczytuje dane i kompletuje je w całą paczkę.

4.3 Inicjalizacja tych podwarstw

Inicjalizacja odpowiednich podwarstw odbywa się tylko i wyłącznie w protokole wstępnym. Raz zainicjalizowanych podwarstw nie można wyłączyć.

5 Przepływ komunikatów

Sekcja ta opisuje przepływ komunikatów, a także format każdego z komunikatów. W protokole możemy wyróżnić kilka podprotokołów zależne od stanu w którym połączenie się znajduje. W szczególności bardzo sztywno należy odgrodzić protokół wstępny — w którym odbywa się negocjacja parametrów połączenia i autoryzacja z protokołem właściwym — w którym jest realizowana właściwa funkcjonalności systemu LoXiM, a zatem wykorzystywane są podprotokoły: przeprowadzania zapytań, przesyłania komunikatu asynchronicznego i kończenia połączenia.

5.1 Nazewnictwo paczek

Na potrzeby tego dokumentu przyjmujemy następujący schemat nazywania paczek:
{znacznik grupy komunikatów}-{znacznik strony, która wysyła tę paczkę}-{identyfikator znaczenia}
Gdzie znacznik grupy paczek może przyjąć następujące wartości:

W — Podprotokół wstępny

Q — Podprotokół przeprowadzania zapytań

V — Podprotokół przesyłania wartości

A — Komunikat asynchroniczny

S — Komunikaty standardowe (np. OK, ERROR)

Znacznikiem strony wysyłającej komunikat może być jedna z następujących możliwości:

C — Tylko klient wysyła tego typu komunikat

S — Tylko serwer wysyła tego typu komunikat

SC — Zarówno klient, jak i serwer mogą wysłać ten komunikat

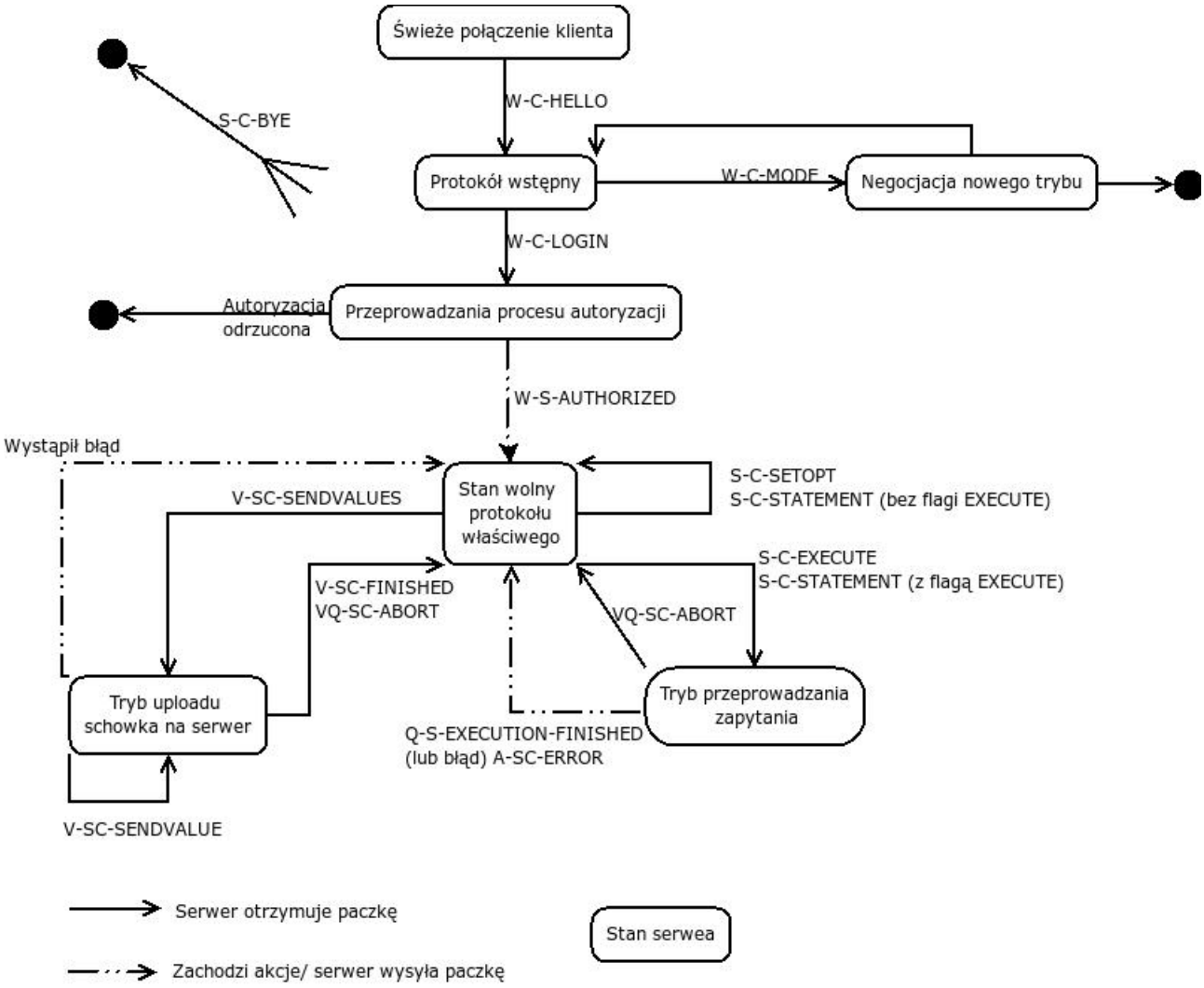
5.2 Metody opisu formatu paczki

Aby ułatwić zapoznawanie się z formatem paczek będziemy je prezentowali w tabeli o następującym formacie:

Od - do	**Typ/Wartość**	**Zawartość**
-5 → -5	10 (uint8)	Identyfikator paczki
-4 → -1	0 (uint32)	Długość właściwej zawartości paczki
0 → ...	wartość (typ)	Opis ogólny pole

Należy zwrócić uwagę, że będziemy prezentowali offsety względem segmentu danych.

5.3 Stany serwera



Rysunek 3: Diagram przejść i stanów serwera

5.4 Protokół wstępny

Protokół wstępny służy do negocjacji parametrów połączenia (kto, do której bazy danych, czy z szyfrowaniem i kompresją, jakie są możliwości serwera i klienta), a także do przeprowadzenia autoryzacji użytkownika.

Ogólny schemat konwersacji jest następujący:

1. Klient nawiązuje połączenie (w przypadku TCP otwiera odpowiedni port na odpowiednim komputerze)
2. Serwer (o ile nie jest skonfigurowana polityka odrzucania połączeń z hosta klienta - raczej na zaporze sieciowej) przyjmuje (akceptuje) połączenie i nic nie wysyła.
3. Klient wysyła paczkę W-C-HELLO (patrz: 5.4.1) — klient ujawnia, że chce rozmawiać z LoXiM'em po tym protokole.
4. Serwer odpowiada paczką W-S-HELLO (patrz: 5.4.2) — serwer ujawnia swoje cechy i możliwości.
5. Klient ewentualnie (kilkukrotnie) wysyła paczkę W-C-MODE (patrz: 5.4.3) — w którym prosi serwer o zmianę trybu (włączenie kompresji, szyfrowania). Odbywa się stosowna konwersacja związana z zamówionymi podwarstwami (jeśli serwer je obsługuje — np. SSL handshake). Serwer potwierdza przyjęcie zlecenia wykonania tej operacji S-SC-OK (patrz: 5.8.1) lub zgłasza błąd S-SC-ERROR (patrz: 5.8.2). Następnie

może zostać przesłany zestaw paczek negocjujących nowy tryb. Następny komunikat jest transportowany już w nowym trybie.

6. Klient ewentualnie ustala pożądane cechy połączenia S-C-SETOPT (patrz: 5.8.4)
7. Serwer potwierdza, bądź odrzuca ich przyjęcie.
8. Klient wybiera protokół logowania i informuje o nim serwer W-C-LOGIN (patrz: 5.4.4). Serwer przeprowadza konwersację autoryzującą z klientem według jednej z metod. Podstawowe metody autentykacji opisane są w dalszej części tego dokumentu (patrz: 7.6).
9. Ostatecznie serwer bądź informuje klienta o zatwierdzeniu autoryzacji W-S-AUTHORIZED (patrz: 5.4.5) bądź odrzuca ją i zamyka połączenie S-SC-ERROR (patrz: 5.8.2).
10. Protokół wstępny zostaje zakończony.

5.4.1 W-C-HELLO

Paczka służy nawiązaniu właściwego połączenia na poziomie logicznym pomiędzy klientem, a serwerem. Stanowi formę przedstawienia, dzięki której serwer nabywa przekonanie, że ma do czynienia z uczciwym klientem, a nie skanerem portów, a także przedstawia serwerowi dane, które mogą być przydatne — głównie do celów administracyjnych (śledzenie stanu serwera, diagnostyka).

Można wyróżnić następujące grupy pól w tym komunikacie

Cechy procesu klienta: PID, nazwa, wersja, hostname Służą one umożliwieniu śledzenia akcji mających miejsce aktualnie na serwerze oraz sporządzanie statystyk i logów dotyczących aktywności pojedynczej aplikacji klienta lub pojedynczej maszyny klienckiej. Podawanie PIDu i hostname'a umożliwia np. administratorowi zatrzymanie konkretnego procesu, który obciąża zbytnio serwer.

Oczywiście — jak zawsze, ale tu szczególnie — w żaden sposób nie należy informacjom podawanym tutaj ufać. Zatem np. decyzję o 'dostępnych' metodach logowania serwer powinien podejmować na podstawie adresu IP pobranego z danych połączenia — a nie na podstawie hostname'a przesłanego w tym pakiecie.

Cechy regionalne: Strefa czasowa Strefa czasowa będzie domyślną strefą w której serwer będzie podawał i przyjmował godziny i daty (gdy są one pozbawione informacji o strefie, której dotyczą).

W tym przypadku ponownie (z konieczności — (patrz: 3.6.1)) posługujemy się strefą czasową zapisaną w postaci różnicy czasu między czasem lokalnym, a czasem GMT.

Cechy regionalne: Porównywanie napisów (collation) Pole „collation" służy przekazaniu informacji o obowiązującej metodzie porównywania napisów. W obecnej wersji systemu LoXiM pole to nie będzie wykorzystywane, ale już zapewniamy na jago potrzeby 64bity (np. pierwsze 32 bity mogą posłużyć do określenia języka, który stosujemy, a drugie 32 bity mogą być wykorzystywane na flagi (np. czy $A > a$, czy może $A = a$)). W szczególności przekazując te dane (język i zestaw flag) do algorytmu „Unicode Technical Standard #10, Unicode Collation Algorithm" [5], można otrzymać zasady porównywania napisów odpowiednie dla wybranego języka.

Z algorytmu UTS wynika, że drugie 32 bity można wykorzystać w następujący sposób do jego parametryzowania (podajemy numery najmniej znaczących bitów od 0 do 31:

0-3 — Te 4 bity wykorzystujemy do przechowania własności „Level", czyli liczby cech uwzględnianych przy sortowaniu. UTS przewiduje następujące 5 poziomów:

- **0** — zasada domyślna dla języka
- **1** — L1 (Base letters)
- **2** — L2 (L1+Accents)
- **3** — L3 (L2+Case)
- **4** — L4 (L3+Punct)
- **5** — L5 (L4+Codepoint)

4-5 — Te dwa bity wykorzystamy do przechowania zachowania względem porównywania znaków wielkich i małych. Proponujemy by:

- **0** — Nie wymuszaj (domyślnie dla języka)
- **1** — Uznawaj wielkie i małe litery za tożsame
- **2** — Wielkie litery pierwsze
- **3** — Małe litery pierwsze

6 Jeśli bit zapalony, to używamy opcji „French accents"

7 Jeśli bit zapalony, to używamy opcji „Add case Level"

8 Jeśli bit zapalony, to używamy opcji „Full normalization mode"

9 Jeśli bit zapalony, to używamy opcji „Add Hiragana Level"

10 Jeśli bit zapalony, to używamy opcji „Numeric Collation"

11-31 aktualnie nie używane

Cechy regionalne: Język klienta Język klienta służy tylko umożliwieniu przesyłania komunikatów (takich jak komunikaty o błędach) w języku możliwie bliskim językowi użytkownika. Jeśli język wybrany tą opcją nie jest wspierany, to system wybiera inny — możliwie bliski wybranemu lub domyślny (np. angielski).

Kodowanie znaków nie jest przesyłane — ze względu na założenie, że wszelka komunikacja tekstowa będzie prowadzona za pomocą kodowania UTF-8 (Big-endian). Zatem zawsze do klienta należy przekodowanie takiego napisu z i na stronę kodową klienta.

Głębszych rozważań na poziomie implementacji serwera bazy danych wymaga kwestia traktowania danych regionalnych takich jak strefa czasowa, czy metoda porównywania napisów.

Niektóre serwery baz danych dokonują w różny sposób konwersji danych zawartych w bazie danych na czas lokalny "sesji" użytkownika. Protokół zaleca, by takie zachowanie podlegało konfiguracji za pomocą opcji obsługiwanych przez pakiet S-C-SETOPT (patrz: 5.8.4).

Podobną kwestią jest to, na jakim poziomie baza danych powinna pamiętać metodę porównywania napisów. Relacyjne bazy danych takie jak MySQL i MSSQL pamiętają ją na poziomie kolumny w tabeli), co jest dość elastycznym rozwiązaniem (choć czasem zaskakuje programistę, gdy odkrywa, że przeszukiwania po jednej kolumnie są czułe na wielkość znaków, a po drugiej kolumnie w tej samej tabeli nie są). Występującym, ale uzasadnionym problemem w tym rozwiązaniu jest to, że nie można porównywać wartości w dwóch kolumnach o różnych metodach porównywaniu napisów (w tej sytuacji wzorowa baza danych powinna udostępnić mechanizm specyfikacji według której kolacji to porównanie ma działać — z czym w praktyce się jeszcze nie spotkałem).

Brak schematu w modelu AS_0 bazy Loxim uniemożliwia takie zapamiętywanie informacji o „collation". Zapamiętywanie tej informacji z każdym napisem wydaje się być skrajnie niepraktyczne (duży koszt pamięciowy i niska potencjalna użyteczność). Za podporządkowaniem metody porównywania napisów do ustawień sesji przemawia sytuacja, gdy użytkownik przyzwyczajony do języka np. szwedzkiego prosi bazę danych o przygotowaniu mu posortowanej listy jego międzynarodowych kontrahentów i oczekuje, że dostanie tę bazę posortowaną według wytycznych jego języka. Z analogicznym roszczeniem do tej bazy może wystąpić Niemiec — dla którego naturalne zasady sortowania są nieco inne. Dlatego w zależności od ustawienia aplikacji klienckiej (a zatem sesji) powinna być wybrana metoda porównywania napisów.

Z drugiej strony — nie jest rzeczą właściwą, gdy zapytanie (Emp where title='Programmer') da nam różne wyniki w zależności od ustawień naszej aplikacji klienckiej (np. w jednym przypadku z uwzględnieniem wielkości znaków, a w drugim bez uwzględnienia wielkości znaków). Pewnym rozwiązaniem w tym przypadku jest specyfikowanie metody na poziomie np. korzeni (rootów).

Protokół nie specyfikuje, które rozwiązanie jest słuszne, ale udostępnia pole, które może być wykorzystane do przesłania tych danych. Także przy zastosowaniu zarówno jednego jak i drugiego rozwiązania pole to może być źródłem danych dla odpowiedniej konfiguracji nowo tworzonych obiektów.

Format pakietu W-C-Hello jest następujący:

Od - do	**Typ/Wartość**	**Zawartość**
-5 → -5	10 (uint8)	Identyfikator paczki
-4 → -1	a (uint32)	Długość właściwej zawartości paczki
0 → 7	sint64	PID procesu klienta — może być 0, gdy nie obsługiwany
8 → k	sstring	Nazwa programu klienckiego.
$k+1 \rightarrow l$	sstring	Wersja programu klienckiego.
$l+1 \rightarrow m$	sstring	Hostname — nazwa komputera (z domeną).
$m+1 \rightarrow m+3$	sstring (3 znaki)	Język użytkownika — kod literowy według standardu ISO-639-2 (http://www.loc.gov/standards/iso639-2/php/code_list.php)
$m+4 \rightarrow m+11$	uint64	Kolacji
$m+12 \rightarrow m+12$	sint8	Strefa czasowa klienta w postaci liczby całkowitej opisującej przesunięcie względem GMT. Czyli dopuszczalne wartości to od -14 (to nie jest błąd) do +12.

5.4.2 W-S-HELLO

Paczka służy przedstawieniu możliwości protokołu i podstawowych informacji o serwerze.

Zatem jego format jest następujący:

Od - do	Typ/Wartość	Zawartość
-5 → -5	11 (uint8)	Identyfikator paczki
-4 → -1	*a* (uint32)	Długość właściwej zawartości paczki
0 → 0	p_major=2(uint8)	Numer główny (major) wersji protokołu
1 → 1	p_minor=0(uint8)	Numer pomocniczy (minor) wersji protokołu
2 → 2	s_major (uint8)	Numer główny (major) wersji systemu
3 → 3	s_minor (uint8)	Numer pomocniczy (minor) wersji systemu
4 → 7	max_package_size (uint32)	Rozmiar maksymalnego paczki przesyłanego tym protokołem — powinno być > 1024 i wartość powyżej 1048586 powinna być istotnie uzasadniona (wartość jest odczytywana z konfiguracji serwera)
8 → 15	features (uint64)	Mapa bitowa dostępnych cech serwer'a
16 → 23	auth_methods (uint64)	Mapa bitowa dostępnych metod autoryzacji
24 → 43	salt (char[20])	160 bitowy ciąg losowy — używany przez niektóre metody autoryzacji

Dostępne features dzielą się na:

Tryby transmisji i działania protokołu: 0x0001=F_SSL — połączenie może być szyfrowane metodą SSL

0x0002=F_O_SSL — obligatoryjne połączenie szyfrowane metodą SSL (wymusza też obecność flagi F_SSL)

0x0004=F_ZLIB — połączenie może być kompresowane za pomocą biblioteki ZLIB

Tryb przetwarzania zapytania: 0x0010=F_AUTOCOMMIT — serwer udostępnia tryb autocommit (każde zapytanie zadane poza transakcją rozpoczyna swoją transakcję, które jest automatycznie zamykana po wykonaniu polecenia)

0x0020=F_OPTIMALIZATION — można włączyć optymalizator zapytań

. . . np. poziomy izolacji transakcji

Aktualnie przewidziane metody autentykacji to auth_methods to: (patrz: 7.6)

0x0001=AM_TRUST — serwer uwierzy w każdą podaną tożsamość ((patrz: 7.6.1))

0x0002=AM_MYSQL5_AUTH — autoryzacja metodą stosowaną przez serwer ((patrz: 7.6.2)) MySQL5 [1] — w oparciu o przesyłanie i przechowywanie skrótu hasła algorytmem SHA1. Generalnie schemat jest następujący:

Serwer przechowuje: $SHA1(password)$

Klient przesyła: $SHA1(password) XOR (SHA1(salt.SHA1(SHA1(password))))$

5.4.3 W-C-MODE

Jest to paczka służąca do przejścia w protokole wstępnym na inny tryb transmisji. Obecnie obejmuje to koncepcję szyfrowania, kompresji, a także ewentualnie transmisji w formacie XML.

Jedynym parametrem paczki jest wybrany tryb transmisji. Zatem format paczki jest następujący:

Od - do	Typ/Wartość	Zawartość
-5 → -5	12 (uint8)	Identyfikator tej paczki
-4 → -1	8 (uint32)	Długość właściwej zawartości paczki
0 → 7	nowy_tryb (uint64)	JEDNA ze stałych trybu transmisji:

Przewidywane tryby transmisji to:

TT_SSL=1 komunikacja szyfrowana z wykorzystaniem protokołu SSL

TT_ZLIB=2 komunikacja kompresowana

W kolejne tryby należy wchodzić pojedynczo. Nowy tryb umieszczany jest na szczycie stosu (warstwa transportu znajduje się na spodzie stosu). Zatem, aby sensownie uruchomić jednocześnie kompresję i szyfrować, należy najpierw uruchomić szyfrowanie, a później kompresję. W ten sposób dane zostaną najpierw skompresowane, a później zaszyfrowane.

Dostępne odpowiedzi to:

S-SC-OK Polecenie zostało zaakceptowane. Za chwile nastąpią negocjacje w kwestii ustalenia nowego trybu. Szczegóły ustalenia konkretnego trybu zależą od wybranego trybu. Strona, która inicjalizuje dalsza konwersację także zależy od konkretnego elementu (np. w przypadku SSL'u serwer powinien rozpocząć „HANDSHAKE").

Jeśli proces negocjacji nowego trybu zakończy się porażką, to połączenie musi zostać natychmiast zerwane.

S-SC-ERROR Nie jest możliwe przejście to żądanego trybu z jakiegoś powodu. Spodziewane komunikaty odpowiedzi to może być:

ERR-ModeNotAvoilable ModeNotAvoilable.

ERR-ModeAlredySet ModeAlredySet.

ERR-Internal Internal.

5.4.4 W-C-LOGIN

Komunikat służy przekazaniu przez klienta serwerowi informacji o tym, za pomocą którego mechanizmu logowania klient będzie się chciał zalogować do serwera. W praktyce jedynym parametrem paczki jest identyfikator wybranej metody autoryzacji 7.6). Zatem format paczki jest następujący:

Od - do	Typ/Wartość	Zawartość
-5 → -5	13 (uint8)	Identyfikator paczki login
-4 → -1	8 (uint32)	Długość właściwej zawartości paczki
0 → 7	nowy_tryb (uint64)	JEDNA z wartości metody autoryzacji (patrz: 7.6)

W momencie wysłania tej paczki — zarządzanie protokołem jest oddane procesowi realizującemu konkretną metodę autoryzacji. Metoda taka powinna zapewnić to, że serwer po jej realizacji wyśle pakiet W-S-AUTHORIZED lub zgłosi błąd i zakończy połączenia, a klient po zakończeniu działanie tej metody, będzie wiedział, że ona się zakończyła i będzie gotowy na odbiór pakietu W-S-AUTHORIZED lub informacji o błędzie.

W sytuacji, gdy serwer otrzyma paczkę W-C-LOGIN chcącą przeprowadzić autoryzację metodą nie wspieraną przez serwer — serwer powinien natychmiast zerwać połączenie.

5.4.5 W-S-AUTHORIZED

Jest to paczka potwierdzająca skuteczną autoryzacje i oświadczający o zakończeniu pracy serwera w trybie wstępnym i o mówiący przejściu w tryb pracy właściwej.

Format jest następujący:

Od - do	Typ/Wartość	Zawartość
-5 → -5	14 (uint8)	Identyfikator tej paczki
-4 → -1	0 (uint32)	Długość właściwej zawartości paczki

5.4.6 W-C-PASSWORD

Jest to paczka w którym klient autoryzujący się przy pomocy hasła powinien przesłać swój login i hasło. W szczególności ta paczka wykorzystuje metody autoryzacji: Trust oraz MySQLpassword (patrz: 7.6).

Format jego jest następujący:

Od - do	Typ/Wartość	Zawartość
$-5 \rightarrow -5$	15 (uint8)	Identyfikator tej paczki
$-4 \rightarrow -1$	m (uint32)	Długość właściwej zawartości tego paczki
$0 \rightarrow n$	login (sstring)	Login autoryzującego się użytkownika
$n+1 \rightarrow m-1$	password (bytes)	Hasło lub jego skrót lub NULL w przypadku metody Trust

5.5 Protokół właściwy — obsługa zapytania

5.5.1 Q-C-STATEMENT

Paczka służy do przesłania zapytania na serwer. Jeżeli flaga EXECUTE jest zapalona — to znaczy, że przesyłamy proste — bezparametrowe zapytanie — które chcemy by zostało natychmiast wykonane przez serwer. W przeciwnym przypadku przesyłamy tylko zapytanie — by zostało sparsowany — i by został my nadany numer StetementId. Posługując się później tym numerem będziemy mogli wielokrotnie zbindować parametry do zapytania i je wykonać (Q-C-EXECUTE (patrz: 5.5.3)).

Format paczki jest następujący:

Od - do	Typ/Wartość	Zawartość
$-5 \rightarrow -5$	64 (uint8)	Identyfikator tej paczki
$-4 \rightarrow -1$	m (uint32)	Długość właściwej zawartości tej paczki
$0 \rightarrow 7$	flags (uint64)	Flagi ustawiające opcje zapytania (patrz: dostępne flagi poniżej)
$8 \rightarrow m-1$	statement (string)	Zapytanie wysyłane na serwer

Aktualnie przewidziane flagi to:

EXECUTE = 0x0001 — jeśli flaga jest zapalona to zapytanie zachowuje się tak, jakby natychmiast po nim została wysłana Q-C-EXECUTE (patrz: 5.5.3). Czyli zamiast odpowiedzi Q-S-STMTPARSED (patrz: 5.5.2) pojawi się albo odpowiedź Q-S-EXECUTING (patrz: 5.5.4), albo zostanie zgłoszony któryś z błędów charakterystycznych dla poleceń Q-C-STATEMENT (patrz: 5.5.1) i Q-C-EXECUTE (patrz: 5.5.3).

READONLY = 0x0002 — jeśli flaga jest zapalona to zapytanie nie ma prawa wprowadzać żadnych modyfikacji w bazie danych

W odpowiedzi na tę paczkę może przyjść paczka Q-S-STMTPARSED (patrz: 5.5.2) (jeśli nie była podniesiona flaga EXECUTE) albo (Q-S-EXECUTING (patrz: 5.5.4) — jeśli była podniesiona flaga EXECUTE) albo jeden z poniższych błędów:

ERR-SyntaxError SyntaxError.

ERR-OperationNotAllowed OperationNotAllowed.

ERR-Internal Internal.

oraz błędy charakterystyczne dla Q-C-EXECUTE (patrz: 5.5.3) — o ile flaga EXECUTE jest zapalona

5.5.2 Q-S-STMTPARSED

Paczka informuje, że analiza składniowa zapytania się powiodła i, że zapytaniu został nadany StatementId. Paczka jest wysyłana w sposób synchroniczny w odpowiedzi na komunikat Q-C-STATEMENT (patrz: 5.5.1).

Format jest następujący:

Od - do	Typ/Wartość	Zawartość
$-5 \rightarrow -5$	65 (uint8)	Identyfikator tej paczki
$-4 \rightarrow -1$	16 (uint32)	Długość właściwej zawartości tej paczki
$0 \rightarrow 7$	statementId (uint64)	Id nadane temu zapytaniu/poleceniu
$8 \rightarrow 15$	paramsCnt (uint32)	Liczba parametrów do ustalenia w sparsowanym zapytaniu.

5.5.3 Q-C-EXECUTE

Paczkę tę wysyła klient w celu poinformowania serwera o tym, że chce wykonać zadane poprzez statementId zapytanie. Często, także tym zapytaniem będzie się dokonywało bindowania bindowania poszczególnych parametrów zapytania z identyfikatorami wartości w schowku (patrz obsługa schowka) w celu wykonania zapytania.

Format paczki jest następujący:

Od - do	Typ/Wartość	Zawartość
$-5 \rightarrow -5$	66 (uint8)	Identyfikator tej paczki
$-4 \rightarrow -1$	m (uint32)	Długość właściwej zawartości tej paczki
$0 \rightarrow 7$	statementId (uint64)	Id nadane temu zapytaniu/poleceniu
$8 \rightarrow 15$	flags (uint32)	Flagi — wskazówki dotyczące wyników zapytania — patrz niżej
$16 \rightarrow 19$	paramsCnt (uint32)	Liczba parametrów do ustalenia w sparsowanym zapytaniu.
$m_{i-1} \rightarrow m_i - 1$	$valueId_i$ dla $i \in (1..paramsCnt)$ (varuint)	Id będące i-tym parametrem. Id powinno się odnosić do Id wartości znajdującym się w schowku sesji — ustalonym uprzednio poprzez paczkę V-SC-SENDVALUE (patrz: 5.6.2)

Przewidywane flagi to (ciąg dalszy do flag z Q-C-STATEMENT):

0x0100 PREFER-DFS Oznacza, że użytkownik sugeruje by wyniki były przesyłane w kolejności umożliwiającej szybką konstrukcję odpowiedzi metodą DFS. Wyklucza się z PREFER-BFS.

0x0100 PREFER-BSF Oznacza, że użytkownik sugeruje by wyniki były przesyłane w kolejności umożliwiającej szybką konstrukcję odpowiedzi metodą BFS. Wyklucza się z PREFER-DFS.

Paczka w sposób synchroniczny jest związana z odpowiedzią Q-S-EXECUTING (patrz: 5.5.4) lub jednym z następujących błędów:

ERR-ParamsIncomplete ParamsIncomplete.

ERR-NoSuchValueId NoSuchValueId.

ERR-OperationNotPermited OperationNotPermited.

ERR-Internal Internal.

5.5.4 Q-S-EXECUTING

Paczka ta jest synchronicznym powiadomieniem o przyjęciu do przetworzenia danego zapytanie (czyli jest wysyłana w odpowiedzi na pakiety Q-C-EXECUTE (patrz: 5.5.3) oraz Q-C-STATEMENT (patrz: 5.5.1) (z flagą EXECUTE)).

Format pakietu jest następujący:

Od - do	**Typ/Wartość**	**Zawartość**
-5 → -5	67 (uint8)	Identyfikator paczki EXECUTING
-4 → -1	0 (uint32)	Długość właściwej zawartości paczki

Otrzymanie tej paczki świadczy o tym, że klient znalazł się w trybie wykonywania zapytania (tzn. że nie możemy wysłać nowego zapytania do czasu przetworzenia lub anulowania bieżącego zapytania, a także to, że aktualnie otrzymywane pakiety z grupy V-. . . dotyczące wyników zwracanych przez to zapytanie).

5.5.5 Q-S-EXECUTION-FINISHED

Paczka informuje, że bieżące polecenie się zakończyło i ewentualnie informuje o liczbie zmian dokonanych przez to zadanie.

Format pakietu jest następujący:

Od - do	**Typ/Wartość**	**Zawartość**
$-5 \rightarrow -5$	70 (uint8)	Identyfikator paczki operationOk
$-4 \rightarrow -1$	a (uint32)	Długość właściwej zawartości paczki
$0 \rightarrow d-1$	modAtomPointerCnt (varuint)	liczba zmodyfikowanych obiektów atomowych i pointerowych przez to zdanie (jeśli nie jest określona to serwer ma to zwrócić NULL)
$d \rightarrow b-1$	delCnt (varuint)	liczba usuniętych obiektów (jeśli nie jest określona to serwer ma to zwrócić NULL)
$b-1 \rightarrow c$	newRootsCnt (varuint)	liczba utworzonych obiektów korzeniowych (jeśli nie jest określona to serwer ma to zwrócić NULL)
$c-1 \rightarrow a-1$	insertsCnt (varuint)	liczba obiektów wstawionych do obiektów złożonych (również tych nowo utworzonych) (jeśli nie jest określona to serwer ma to zwrócić NULL)

Wysłanie (ale nie umieszczenie w kolejce paczek do wysłania) tej paczki przez serwer powoduje, że wychodzi on z trybu przetwarzania zapytania.

Wysyłanie ilości dokonanych zmian ma ułatwić pracę systemów korzystających z mechanizmu tzw. optymistycznego przetwarzania transakcji przez aplikację kliencką (serwery aplikacyjne i mechanizmy w stylu „hibernate").

5.6 Protokół właściwy — przesyłanie wartości

5.6.1 V-SC-SENDVALUES

Paczka informuje drugą stronę, że rozpoczyna się transfer (zbioru) wartości. Może zawierać informacje o orientacyjnej ilości przesyłanych wyników — by umożliwić wyświetlenie użytkownikowi orientacyjnej informacji o postępie.

Format paczki jest następujący:

Od - do	Typ/Wartość	Zawartość
$-5 \rightarrow -5$	32 (uint8)	Identyfikator paczki operationOk
$-4 \rightarrow -1$	a (uint32)	Długość właściwej zawartości paczki
$0 \rightarrow l$	rootValueId (varuint)	ID korzenia — paczki, która będzie zawierała właściwy obiekt z wartością. Wartość NULL nie jest dozwolona.
$l+1 \rightarrow m$	oBundlesCount (varuint)	Orientacyjne liczba paczek, które zostaną przesłane — NULL — jeśli nie znana
$m+1 \rightarrow n$	oBidCount (varuint)	Orientacyjna liczba obiektów, które zostaną przesłane — NULL — jeśli jeszcze nie określona
$n+1 \rightarrow a-1$	pVidCount (varuint)	Dokładna liczba obiektów, które zostaną przesłane — NULL — jeśli jeszcze nie określona

5.6.2 V-SC-SENDVALUE

Paczka ta służy do przesłania pewnej wartości. Każda wartość posiada swoje ID — generowane przez stronę wysyłającą. ID to będzie służyło do budowania bardziej złożonych wartości poprzez grupowanie w odpowiednich strukturach ID-ków identyfikujący elementy leżące głębiej w hierarchii. Nie ma żadnego wymogu, aby ID do których się odwołujemy były przesłane wcześniej niż obiekt z nich korzystający.

Przez wartość rozumiemy każdą konstrukcję zgodną z konstrukcji wartości podaną w [3] w rozdziale „SBA: Environment Stack in the AS_0 Store Model — Result returned by Queries".

Format tych danych jest następujący:

Od - do	Typ/Wartość	Zawartość
$-5 \rightarrow -5$	33 (uint8)	Identyfikator paczki operationOk
$-4 \rightarrow -1$	a (uint32)	Długość właściwej zawartości paczki
$0 \rightarrow l$	ValueId (varuint)	Identyfikator wartości
$l+1 \rightarrow l+1$	Flags (uint8)	Flagi z informacjami o pakiecie
$l+2 \rightarrow m$	Typ wartości	Jedna ze stałych kodowych — opisanych poniżej
$m+1 \rightarrow a-1$	data (zależne od typu)	Dane właściwe wartości — zależne od typu wskazanego przez poprzednią komórkę

Przewidywane flagi to:

TO-BE-CONTINUED=0x01 Następny pakiet będzie zawierał ciąg dalszy informacji do tego pakietu (w tym pakiecie nie zmieściły się wszystkie dane, które miały być przesłane — np. ze względu na ograniczenie max_package_size). Następny pakiet będzie dotyczył tego samego obiektu (ten sam ValueId), ale będzie zawierał dalsze wartości.

Przewidywane typy proste to (nie wszystkie typy muszą być dostępne dla użytkowników):

Nazwa	Kod	Opis	Rozdzielalny
UINT8	0x0001	Zapisany jako uint8	-
SINT8	0x0002	Zapisany jako sint8	-
UINT16	0x0003	...	-
SINT16	0x0004	...	-
UINT32	0x0005	...	-
SINT32	0x0006	...	-
UINT64	0x0007	...	-
SINT64	0x0008	...	-
BOOL	0x0009	Zapisany jako bool	-
DATE	0x000A	Data	-
TIME	0x000B	Czas bez strefy czasowej	-
DATETIME	0x000C	Data i czas bez strefy czasowej	-
TIMETZ	0x000D	Czas ze strefą czasową	-
DATETIMETZ	0x000E	Data i czas ze strefą czasową.	-
BYTES	0x000F	Obiekt binarny (nie musi być długi)	+
VARCHAR	0x0010	Tekst (zakodowany w UTF-8 Big-endian).	+
DOUBLE	0x0011	Dane rzeczywiste podwójnej precyzji	-

Przewidywane typy organizujące strukturę (Zgodnie z [3] „SBA: Enviroment Stack in the AS_0 Store Model — Results returned by queries"):

Nazwa	Kod	Opis	Rozdzielalny
VOID	0x0080	Typ pusty.	-
LINK	0x0081	Taka wartość jak opisana poprzez pakiet o danym valueId	-
BINDING	0x0082	Związanie poprzez daną nazwę danej wartości	-
STRUCT	0x0083	Struktura elementów (nie stanowiąca kolekcji)	+
BAG	0x0084	Kolekcja będąca multizbiorem elementów	+
SEQUENCE	0x0085	Kolekcja będąca ciągiem elementów	+
REF	0x0086	Odwołanie do pewnego miejsca w strukturze danych (najprawdopodobniej poza przesłanym pakietem), raczej do krótkotrwałych operacji	-
EXTERNAL_REF	0x0087	Odwołanie do pewnego miejsca w strukturze danych (najprawdopodobniej poza przesłanym pakietem), w celu w przechowywania w innym zewnętrznym systemie	-

W celu poznania struktury poszczególnych formatów danych, należy zapoznać się z rozdziałem 6.

5.6.3 V-SC-FINISHED

Paczka informuje stronę odbierającą, że wszystkie wartości, które miały zostać przesłane — zostały już wysłane. Oczekuje, że strona odbierająca sprawdzi spójność danych i odpowie synchronicznie, bądź poprzez S-SC-OK (patrz: 5.8.1) bądź poprzez zgłoszenie błędu. W obu sytuacjach jednak wysłanie tej paczki powoduje zakończenie tryby przesyłania wyniku.

Format paczki jest następujący:

Od - do	Typ/Wartość	Zawartość
-5 → -5	34 (uint8)	Identyfikator paczki
-4 → -1	0 (uint32)	Długość właściwej zawartości paczki

5.6.4 V-SC-ABORT

Paczka może być wysłana bądź przez stronę odbierającą, bądź przez wysyłającą. Oznacza ona, że należy zaprzestać wysyłać lub oczekiwać na dane.

Paczka ta może być także wysłana poza trybem przesyłania wartości przez serwer - i informuje wtedy o przerwaniu wykonywania (obsługi) bieżącego zapytania.

Wysłanie jej w trybie wykonywania zapytania powoduje opuszczenie tego trybu.

Format paczki jest następujący:

Od - do	Typ/Wartość	Zawartość
$-5 \rightarrow -5$	35 (uint8)	Identyfikator paczki
$-4 \rightarrow -1$	n (uint32)	Długość właściwej zawartości paczki
$0 \rightarrow 3$	reasonCode (uint32)	Kod błędu mówiący o przyczynie przerwania zapytania, 0 — brak uzasadnienia
$4 \rightarrow n-1$	reasonString (sstring)	Słowny opis powodu (najlepiej w języku określonym w fazie wstępnej)

Przewidywane obecnie reasonCody to:

ADMINISTRATION-REASON Twoje zapytanie zostało przerwane na skutek działań administracyjnych

YOU-ARE-TRANSACTION-VICTIM Twoja transakcja została wycofana ze względu na powstanie zastoju (DEADLOCK)

OPERATION-NOT-PERMITED Próba wykonania niedozwolonej operacji

TIME-LIMIT-EXCEEDED Przekroczony maksymalny czas wykonywania zapytania

OUT-OF-MEMORY Zabrakło pamięci na dalsze przetwarzanie wyniku

TYPE-CHECK-ERROR Problem konwersji (dynamicznej kontroli typów)

OTHER-RUN-TIME-ERROR Inny błąd wykonania

5.7 Protokół właściwy — paczki różne

5.7.1 A-SC-PING

Paczka jest wysyłany okresowo przez serwer do klienta (lub potencjalnie przez klienta do serwera) w celu stwierdzenia, czy łączność z klientem jest zachowana, a klient się nie zawiesił. Paczka składa się w praktyce tylko z nagłówka.

Zatem jego format jest następujący:

Od - do	Typ/Wartość	Zawartość
-5 → -5	128 (uint8)	Identyfikator paczki ping
-4 → -1	0 (uint32)	Długość właściwej zawartości paczki

Paczka A-SC-PING nie jest obsługiwany w fazie wstępnej. Tam połączenie zostanie automatycznie zamknięte przez serwer po upływie login-timeout.

5.7.2 A-SC-PONG

Paczka wysyłany w odpowiedzi na paczkę A-SC-PING (patrz: 5.7.1).

Paczka składa się w praktyce tylko z nagłówka.

Zatem jego format jest następujący:

Od - do	Typ/Wartość	Zawartość
-5 → -5	129 (uint8)	Identyfikator paczki pong
-1 → -4	0 (uint32)	Długość właściwej zawartości paczki

Paczka A-SC-PONG (patrz: 5.7.2) nie jest obsługiwany w fazie wstępnej. Tam połączenie zostanie automatycznie zamknięte przez serwer po upływie login-timeout.

5.8 Komunikaty ogólne

5.8.1 A-SC-OK

Paczka potwierdzający przyjęcie i zrozumienie komendy wysłanej na serwer przez klienta. Paczka składa się w praktyce tylko z nagłówka.

Zatem jego format jest następujący:

Od - do	Typ/Wartość	Zawartość
-5 → -5	01 (uint8)	Identyfikator paczki ok
-4 → -1	0 (uint32)	Długość właściwej zawartości paczki

5.8.2 A-SC-ERROR

Format paczki jest następujący:

Od - do	Typ/Wartość	Zawartość
-5 → -5	02 (uint8)	Identyfikator paczki error
-4 → -1	$12 + n$ (uint32)	Długość właściwej zawartości paczki
0 → 3	uint32	Kod błędu
4 → a	varuint	Numer jednostki wykonywania (np. StatementId lub Portalu lub NULL) — z którą związany jest błąd
$a + 1 \rightarrow n$	sstring	Opis błędu — słowny
$n + 1 \rightarrow n + 4$	uint32	Numer linii w której błąd wystąpił (0 — gdy nie dotyczy)
$n + 5 \rightarrow n + 8$	uint32	Numer znaku w linii w której błąd wystąpił (0 — gdy nie dotyczy)

5.8.3 A-SC-BYE

Paczka zostaje wysyłana, by poinformować drugą stronę o zakończeniu połączenia. Powinna zostać wysłana, gdy w tym protokole jest napisane, że strona może zakończyć połączenie.

Strona, która otrzymuję tę paczkę powinna zamknąć połączenie po jej odczytaniu (bez wysyłania zwrotnego A-SC-Bye).

Jeśli jest napisane, że strona może/musi zerwać połączenie to ta paczka nie może zostać wysłana, tylko połączenie musi zostać zamknięte na poziomie TCP/IP.

Format paczki jest następujący:

Od - do	Typ/Wartość	Zawartość
-5 → -5	03 (uint8)	Identyfikator paczki bye
-4 → -1	0 (uint32)	Długość właściwej zawartości paczki
0 → n	reason (string)	Powód zakończenia połączenia — do celów informacyjnych. Może być NULL.

5.8.4 S-C-SETOPT

W protokole wstępnym ta paczka służy do ustawienia opcji przez klienta, które mogą być istotne w procesie stwierdzania, czy klient ma dostęp dla danego zasobu, czy go niema. .

W praktyce polecenie to przesyła parę dwóch stringów: klucz i wartość. Zatem konstrukcja paczki jest następująca:

Od - do	Typ/Wartość	Zawartość
$-5 \rightarrow -5$	130 (uint8)	Identyfikator paczki setopt
$-4 \rightarrow -1$	8 (uint32)	Długość właściwej zawartości paczki
$0 \rightarrow n$	key (sstring)	Klucz
$n+1 \rightarrow m$	value (string)	Wartość nadana kluczowi

Z punktu widzenia LoXiM'a sensownym się wydaje wprowadzenie w protokole wstępnym następujących parametrów:

local_root Wskazuje, który obiekt stanowi root'a dla tego połączenia, a zatem wszystkie łączenia (bindings) znajdujące się w środowisku początkowej ewaluacji SBQL'a będą pochodziły z niego. Coś w stylu Unikowego polecenia 'chroot'.

W protokole właściwym natomiast:

autocommit Dopuszczalne wartości to 'true' i 'false'. Ustawia opcję autocommit.

Dygresja: Wydaje się sensownym zastąpienie w protokole właściwym tego polecenia poprzez czystego SBQL'a odwołującego się do wirtualnych wpisów w bazie danych.

Można sobie wyobrazić następujące zapytanie SBQL: $$.session.autocommit:=true, gdzie $$ jest wirtualnym korzeniem (coś na podobieństwo Unix'owego katalogu /proc)

6 Złożone typy danych

6.1 VOID

Dane tego typu paczki nie istnieją (ich długość wynosi 0).

6.2 LINK

Zawiera informację, że wartość ta jest opisana w innej paczce o zadanym ID.

Od - do	**Typ/Wartość**	**Zawartość**
$0 \rightarrow$ a	*valueId* (varuint)	Identyfikator paczki w której jest właściwa wartość

Ten typ będzie używany przeważnie w innych (poniższych) złożonych strukturach w celu uniknięcia rekurencyjnego budowania „dużych pakietów".

6.3 BINDING

Jest to związanie pewnej wartości poprzez konkretną nazwę wewnątrz obiektu.

Od - do	**Typ/Wartość**	**Zawartość**
$1 \rightarrow a$	*bindingName* (sstring)	Nazwa z którą wiążemy daną wartość
$a+1 \rightarrow b$	*type* (varuint)	Typ wartości do którego jest binding
$b+1 \rightarrow c$	(format zależy od *type*)	Stosowna wartość (format zależny od typu)

FEATURE: W następnych wersjach można rozważyć kompresje powtarzających się bindingów (by nie przechowywać ich jako "string" ale jako słownik bindingów i przesyłać tylko id binding'a)

Zatem propozycja alternatywnego formatu przesyłania bindingów jest taka:

Od - do	**Typ/Wartość**	**Zawartość**
$0 \rightarrow 0$	NULL (sstring)	
$1 \rightarrow a$	*valueId* (varuint)	Identyfikator wcześniej wysłanego (w tej grupie wartości) bindingu
$a+1 \rightarrow b$	*type* (varuint)	Typ wartości do którego jest binding
$b+1 \rightarrow c$	(format zależy od *type*)	Stosowna wartość (format zależny od typu)

Oba te formaty są ze sobą zgodne. Jeśli dany string ma wartość NULL, to znaczy, że mamy do czynienia z drugim formatem.

6.4 STRUCT, BAG, SEQUENCE

Wszystkie te trzy formaty danych "póki co" mają ten sam format.

Przewidujemy dwie możliwości zapisywania takich struktur — jednorodną i różnorodną. Celem wprowadzenia formatu jednorodnego jest uniknięcie przesyłania przy każdym elemencie zbioru informacji o jego typie — poprzez podanie jednego globalnego typu dla wszystkich elementów. Formę jednorodną wyróżnia nie null'owy „typ globalny".

Zatem format typu jednorodnego to:

Od - do	Typ/Wartość	Zawartość
$0 \rightarrow a$	count (varuint)	liczba wartości zawartych **w tym pakiecie**
$a+1 \rightarrow b$	*globalType* (varuint)	Typ wszystkich elementów kolekcji
$b+1 \rightarrow c_1$	(zależny od „globalType")	Wartość 1-wsza
$c_1+1 \rightarrow c_2$	(zależny od „globalType")	Wartość 2-ga
$\ldots \rightarrow \ldots$	$\ldots$	$\ldots$
$c_{count}+1 \rightarrow c_{count+1}$	(zależny od „globalType")	Wartość ostatnia

Analogicznie format pakietu różnorodnego uzyskujemy poprzez przesunięcie pole "globalType" do każdej pojedynczej wartości:

Od - do	Typ/Wartość	Zawartość
$0 \rightarrow a$	count (varuint)	liczba wartości zawartych **w tym pakiecie**
$a+1 \rightarrow b$	*NULL* (varuint)	Typ wszystkich elementów kolekcji
$t_1 \rightarrow c_1-1$	$type_1$ (varuint)	Typ wartości 1-wszej
$c_1 \rightarrow t_2-1$	(zależny od „$type_1$")	Wartość 1-wsza
$t_2 \rightarrow c_2-1$	$type_2$ (varuint)	Typ wartości 2-giej
$c_2 \rightarrow t_3-1$	(zależny od „$type_2$")	Wartość 2-ga
$\ldots \rightarrow \ldots$	$\ldots$	$\ldots$
$t_{count}+1 \rightarrow c_{count}-1$	$type_{count}$	Typ ostatniej wartości
$c_{count} \rightarrow c_{count+1}$	(zależny od „$type_{count}$")	Wartość ostatnia

Pole tego typu ten może być rozłożony na wiele pakietów przesyłających wartość — poprzez zaznaczenie flagi TO_BE_CONTINUED i przesłanie identycznego nagłówka wartości dla wszystkich pakietów).

6.5 REFERENCE

Jest to typ zawierający pewną referencję do innego miejsca w strukturze danych przechowywanej w bazie danych. Jest on po prostu wewnętrznym identyfikatorem bazy danych i nie powinien być interpretowany przez program kliencki.

Więc format jest następujący:

Od - do	Typ/Wartość	Zawartość
$0 \rightarrow 7$	*valueId* (uint64)	Wewnętrzny identyfikator miejsca

6.6 EXT_REFERENCE

Jest to typ już zarezerwowany, ale jeszcze nie wprowadzony. Służy do, oprócz trzymania wewnętrznego identyfikatora w systemie LoXiM, przechowywanie stempla czasowego — świadczącego o czasie — w którym podany

identyfikator był stworzony. Dzięki temu będzie możliwe po stronie systemu bazy danych powtórne wykorzystywanie identyfikatorów.

Zatem wstępnie jego format ustalamy w następujący sposób:

Od - do	**Typ/Wartość**	**Zawartość**
$0 \rightarrow 7$	*valueId* (uint64)	Wewnętrzny identyfikator miejsca
$8 \rightarrow 15$	*stamp* (uint64)	Stempel czasowy lub coś w tym stylu. Ale implementacja klienta i tak nie musi tego interpretować — a nawet nie powinna.

7 Bezpieczeństwo

Protokół sieciowy jest z pewnością najbardziej kuszącą częścią systemu — z punktu widzenia osoby chcącej naruszyć jego bezpieczeństwo.

7.1 Całkowity brak zaufania

Należy zachować całkowity brak zaufania w kwestiach poprawności przesyłanych danych.

Pod żadnym pozorem, serwer (ani żadna inna aplikacja odczytująca dane) **NIE MOŻE ZAKŁADAĆ, ŻE DOSTARCZANE DANE SĄ POPRAWNE**.

Długości wszystkich buforów i zakresy wartości wszystkich zmiennych myszą być bardzo dokładnie kontrolowane.

Przy stwierdzeniu naruszenie zasad protokołu przez którąś stronę komunikacji, połączenie powinno zostać natychmiast zerwane, a stosowny komunikat powinien zostać zapisany do logów (serwer) lub udostępniony użytkownikowi systemu (klient).

7.2 Utrudnienia dla skanerów portów

Zgodnie z opisem protokołu wstępnego system stara się ukryć, to że jest serwerem LoXiM do czasu, aż klient dokona zgodnego z tym protokołem powitania (patrz: 5.4.1). Służy to utrudnieniu zdobycia informacji o działających na atakowanym systemie aplikacjach. Dzięki temu skaner portów musi oprócz otworzenia portu, dokonać odpowiedniego zapytania, zanim zostanie poinformowany o opcjach i wersjach protokołu — a zatem o potencjalnych miejscach, o których atakujący może poszukiwać informacji o lukach w zabezpieczeniach.

7.3 Synchroniczność/Asynchroniczność

By zmniejszyć dodatkowo pule potencjalnych zagrożeń, ustalamy, że protokół wstępny jest w pełni synchroniczny. Dopiero w protokole właściwym stają się dostępne komunikaty przesyłane przez serwer do klient w sposób asynchroniczny (nie będące odpowiedzią na żądanie klienta).

7.4 Limity czasów

Można wprowadzić limit czasów na wykonanie różnych operacji przez użytkownika, których celem utrudnić ataki poprzez nawiązywanie zbyt dużej ilości połączeń, które nawet nie zastają autoryzowane, lub które nazbyt długo pozostają nie używane.

authorization timeout — czas pomiędzy nawiązaniem połączenia, a przeprowadzeniem skutecznego logowania. (opcja może zostać wyłączona).

authorization delay — czas jaki serwer odczeka po nieudanej próbie autoryzacji, zanim poinformuje użytkownika o porażce.

idle time — czas, który jeśli upłynie pomiędzy ostatnią merytoryczną wymianą paczek (i kolejka zadań dla danego połączenia po stronie serwera jest pusta) to serwer może zerwać połączenie. (opcja może zostać wyłączona).

7.5 S(B)QL Injection

Jednym z najczęstszych ataków na systemy związane z bazami danych jest atak „SQL injection". Polega on na takim podaniu parametrów dla zapytania, że w wyniku działania oprogramowanie to wygeneruje takie zapytanie, które odniesie inny skutek, niż programista zamierzał. W większości wypadków — poprzez źle wyescapowany tekst - do zapytania dostają się dodatkowe polecenia lub modyfikacje bieżącego zapytania.

Żeby takie działanie utrudnić protokół wprowadza następujące rozwiązania:

1. Tylko jedno polecenie „per paczkę" jest dozwolone. Tzn. że jeżeli w pojedynczym pakiecie zostaną wysłane dwa zapytania (standardowo oddzielone średnikiem) to nie zostanie wykonane z nich żadne).

2. Charakter polecenia: W pakiecie — razem z zapytanemu jest wysyłana informacja, czy jest ono typu 'readonly', czy 'readwrite'. Zapytania readonly nie mają prawa wprowadzić modyfikacji do żadnych obiektów, więc „wstrzyknięte" zapytanie nie dokona nam modyfikacji bazy danych.

Nadal potencjalnie niebezpieczne pozostaną operację zapisujące dane w bazie danych i podatne na sqlinjection, a także będzie istniała możliwość pozyskania dodatkowych danych poprzez wpłynięcie na spectrum wyników zwróconych w zapytaniu.

Należy wspomnieć w tym miejscu także to, że protokół udostępnia mechanizmy parametryzacji zapytań, co nie tylko pozwala uniknąć problemów związanych z wstrzyknięciem złośliwego kodu, ale także pozwala współdzielić plany zapytań przez wiele wywołań zapytania — co pozytywnie wpływa na wydajność.

7.6 Metody autoryzacji

7.6.1 Trust (pełne zaufanie)

Brak metody autoryzacji. W praktyce jej dostępność powinna zupełnie wyłączona, a jej istnieje służy tylko umożliwieniu zmiany hasła administratora po tym, jak ten go zapomni (i to tylko dla połączeń z lokalnego interfejsu sieciowego).

Protokół autoryzacji metodą trust wygląda następująco:

1. Klient wysyła paczkę W-C-PASSWORD (patrz: 5.4.6) z nazwą użytkownika taką jaką chce otrzymać po zalogowaniu i hasłem ustawionym na NULL.
2. Serwer sprawdza, czy taki użytkownik istnieje i jeśli istnieje, to autoryzacja zostaje potwierdzona poprzez wysłanie do klienta komunikatu W-S-AUTHORIZED (patrz: 5.4.5). Serwer przechodzi we właściwy tryb pracy.

   Jeśli natomiast podany użytkownik nie istnieje, to serwer odpowiada błędem ERR-NoSuchUser, po czym zrywa połączenie.

7.6.2 Autoryzacja hasłem — jak w MySQL

Autoryzacja zaszyfrowanym hasłem. Hasło jest także na serwerze przechowywane w postaci zaszyfrowanej.

Protokół autoryzacji metodą MySQLPassword wygląda następująco:

1. Klient pobiera 160 bitów z początku przesłanego przez serwer w komunikacie W-S-HELLO (patrz: 5.4.2) zbioru losowych danych 'salt'.
2. Klient pobiera nazwę użytkownika i hasło. Wylicza następującą daną

   $$SHA1(password) XOR (SHA1(salt . SHA1(SHA1(password))))$$

   .

   UWAGA! Zakładamy, że funkcja SHA1 dla danego tekstu (zakodowanego jako UTF-8 Big-endianbez żadnego znaku końca i znacznika długości) zwraca zbiór 160 bitów (czyli 20 bajtów) — a nie tekst z wartością w hex'ach - które jako wynik zwraca wiele funkcji bibliotecznych.
3. Klient wysyła paczkę W-C-LOGIN (patrz: 5.4.4) z wybraną procedurą autentykacji.
4. Klient wysyła paczkę W-C-PASSWORD (patrz: 5.4.6) z nazwą użytkownika taką jaką chce otrzymać po zalogowaniu i hasłem ustawionym na wyżej wyliczony skrót hasła.
5. Serwer sprawdza, czy taki użytkownik istnieje i jeśli istnieje, to porównuje przesłane hasło z samodzielnie wyliczonym skrótem. to autoryzacja Jeśli wszystko się zgadza — to autoryzacja zostaje potwierdzona poprzez wysłanie do klienta komunikatu W-S-AUTHORIZED (patrz: 5.4.5). Serwer przechodzi we właściwy tryb pracy.

   Jeśli natomiast podany użytkownik nie istnieje, to serwer odpowiada błędem ERR-AccessDenied, po czym zrywa połączenie.

7.7 Limity

Wydaje się celowym wprowadzenie następujących limitów:

Maksymalna liczba użytkowników (połączeń)

Maksymalny rozmiar paczki przesyłanego w protokole

Maksymalny rozmiar schowka na parametry

8 Etapy realizacji projektu

8.1 Faza 1

- Zaimplementowanie transportu wszystkich opisanych w tym dokumencie paczek.
- Zaimplementowanie bibliotek umożliwiających nawiązywanie połączeń.

8.2 Fazy następne

- Wprowadzenie warstwy kompresji (ZLIB)
- Wprowadzenie warstwy komunikacji szyfrowanej (SSL)
- Przesyłanie informacji o typach na potrzeby (pół-)mocnej kontroli typów.
- Autoryzacja za pomocą mechanizmu Kerberos.
- Autoryzacja za pomocą innych mechanizmów uwierzytelniania.

Literatura

[1] MySQL Protocol - version 10. Ian Redfern, [03.12.2006], (http://www.redferni.uklinux.net/mysql/MySQL-Protocol.html)

[2] PostgreSQL Frontend/Backend Protocol - version. PostgreSQL Fundation, [03.12.2006], (http://developer.postgresql.org/pgdocs/postgres/protocol.html)

[3] Stack-Based Approach (SBA) and Stack-Based Query Language (SBQL). Kazimierz Subieta, [03.12.2006], (http://www.sbql.pl)

[4] TDS Protocol Documentation. FreeTDS, [10.12.2006], (http://www.freetds.org/tds.html)

[5] Unicode Technical Standard #10 Unicode Collation Algorithm. Mark Davis, Ken Whistler, [29.05.2008], (http://www.unicode.org/reports/tr10/)

[6] ISO 8601 - Numeric representation of Dates and Time, ISO, [29.05.2008], (http://www.iso.org/iso/support/faqs/faqs_widely_used_standards/widely_used_standards_other/date_and_time_format