STUDIA INFORMATICA 2013
Volume 34 Number 2A (111)

Michał WIDERA
Streams Lab

PROTOKÓŁ KOMUNIKACYJNY STRUMIENIOWEGO SYSTEMU ZARZĄDZANIA DANYMI

Streszczenie. Jedną z funkcji systemu zarządzania danymi jest dostarczenie danych. W środowisku sieciowym najprostszym modelem klienta jest program konsoli komunikującej się za pomocą wybranego protokołu. Z uwagi na konieczność bieżącej aktualizacji odpowiedzi konstrukcja tego protokołu w przypadku rozwijanego systemu odbiega od typowych rozwiązań. W artykule przedstawiono opracowany protokół umożliwiający komunikację sieciową w systemie strumieniowym.

Słowa kluczowe: DSMS, protokół komunikacyjny bazy danych

DATA STREAM MANAGEMENT SYSTEM COMMUNICATION PROTOCOL

Summary. One of the main database management system function is data delivery. Data client in case of network environment could modeled be system console with communication protocol. Considering continuous update of query answer such protocol construction is different from common solutions. This paper presents communication protocol that enables network communication in network system.

Keywords: DSMS, database communication protocol

1. Protokoły komunikacji w systemach przetwarzania danych

Rozważając wymogi konstrukcji dowolnego systemu przetwarzania danych, należy uwzględnić potrzebę i metodę ich prezentacji. W trywialnym przypadku udostępnione dane zostaną zaprezentowane przez system w trybie ciągłym, nawet bez konieczności ich bezpośredniego żądania. Rozważając przypadek bardziej ogólny, musimy zdefiniować właściwy protokół komunikacyjny.

W najbardziej popularnych systemach zarządzania danymi ich podstawowe sieciowe protokoły komunikacyjne nie stanowią jakiegoś szczególnie skomplikowanego przypadku użycia [1]. Są to z reguły protokoły typu pytanie-odpowiedź. Pytanie formułowane jest zgodnie z gramatyką języka zapytań, odpowiedź udzielana jest zazwyczaj w formie tabeli reprezentującej istniejące relacje pomiędzy danymi. Poniżej tej warstwy nie dzieje się też nic niezwykłego. System bazy danych nasłuchuje na wskazanym, powszechnie znanym porcie, a następnie odpowiada, otwierając równocześnie inny port, i umieszcza tam odpowiedź. Taki model komunikacyjny jest powszechnie stosowany również w innych systemach – w szczególności typowa przeglądarka internetowa w podstawowym trybie pracy funkcjonuje podobnie.

Przy rozważaniu problemu od strony bardziej ogólnej, abstrahując od warstw i protokołów sieciowych, można stwierdzić, że system przetwarzający dane może komunikować się za pomocą obszaru pamięci współdzielonej, magistral szeregowych lub innych, jeszcze bardziej uproszczonych metod komunikacji. W szczególnych przypadkach mogą to być nawet bezpośrednie wywołania funkcji lub wywołania funkcji DCOM/CORBA. Jednak poszukując rozwiązań najbardziej wydajnych, nie unikniemy analizy tego zagadnienia w przypadku konieczności umieszczenia systemu zarządzania danymi na zewnętrznej maszynie.

1.1. Szkic wymagań funkcjonalnych strumieniowego systemu zarządzania danymi

Aby właściwie nakreślić przegląd istniejących rozwiązań, w początkowej części artykułu konieczne jest wskazanie właściwego obszaru poszukiwań i zastosowań – krótkiego zestawu wymagań. Zgodnie z założeniami projektowymi [2] system wspiera architekturę typu klient-serwer. Dane napływają z serwera do klientów w określonych z góry odstępach czasu, zapewniając płynność i określoną tolerancję błędów. Jednym z przyjętych mediów komunikacyjnych dla systemu jest sieć komputerowa. Specyficzny sposób wymiany danych pomiędzy systemem zarządzania danymi a klientem wymaga zastosowania lub stworzenia efektywnego i bezpiecznego protokołu komunikacyjnego. Przedstawiony w artykule protokół znajduje zastosowanie w realizacji powyższego celu.

W przypadku rozwijanego systemu całość oprogramowania tworzona jest w języku C++. Przyjęto, że tworzone oprogramowanie będzie przenośne pomiędzy co najmniej dwoma kompilatorami i dwoma systemami. Jako rozszerzenie funkcjonalne języka użyto biblioteki BOOST. Biblioteka ta spełnia wymaganie przenośności pomiędzy kompilatorami i systemami. Tworzony system projektowany jest z myślą o uruchomieniu na platformach typu desktop oraz embedded, tzn. w szczególności pod kontrolą systemu Linux osadzonego na procesorze ARM. Obecnie równolegle prowadzone są prace eksperymentalne nad uruchomieniem systemu na platformie Raspberry PI [3] oraz budżetowych serwerach VPN dostępnych w sieci.

W ramach opracowanego sposobu transmisji danych za pomocą sieci komputerowej pomiędzy klientem a serwerem przewidziano protokół nawiązania połączenia, podobnie jak przedstawiono to w przypadku relacyjnej bazy danych [1]. Odpowiedź serwera wiązała się z dostarczeniem danych atomowych oraz otwarciem strumienia danych jako właściwej odpowiedzi. Pierwsza część komunikacji sieciowej w takim wypadku może zostać zrealizowana za pomocą protokołu z potwierdzeniami, druga część w celu efektywnej realizacji wymaga zastosowania protokołu bez potwierdzeń.

1.2. Przegląd istniejących rozwiązań wspomagających komunikację sieciową

Typowym problemem początkujących twórców oprogramowania jest spontaniczna potrzeba tworzenia idealnych rozwiązań w każdym aspekcie – bez dokładnego oglądania się na to, co robią lub zrobili dotychczas inni twórcy. Perfekcjonizm nie jest niczym złym, jednak bez możliwości wstępnej analizy rozwiązań opisanych i dostępnych na rynku możemy pominąć bardzo ważny i istotny element, dlatego że naszą uwagę skupia ogrom szczegółów. Co więcej, bardzo często zdarza się, że rozwiązania innych twórców dokładnie wypełniają wymagania naszego projektu. Bez przeprowadzenia nawet pobieżnego przeglądu istniejących rozwiązań możemy stworzyć coś, co będzie dowodem braku naszych kompetencji oraz wynikiem straconego czasu.

W analizowanym przypadku uwagę przykuły dwa podstawowe rozwiązania umożliwiające komunikację sieciową. Pierwszym jest biblioteka Thrift [4], drugim – Protocol Buffers [5]. Pierwsze rozwiązanie jest pochodną prac firmy Facebook, drugie zostało stworzone przez firmę Google. Thrift przedstawiono w 2007 roku, obecnie jest projektem *open source* pod opieką Apache Software Foundation. Protocol Buffers został przedstawiony rok później. Podobnie jak Thrift, jest on objęty wolną licencją *open source*.

Obie biblioteki umożliwiają implementację funkcjonalności pod różnymi systemami i językami. W obu przypadkach definiowany jest zestaw struktur danych, które podlegają wymianie. Tworzony protokół w pewnej sformalizowanej formie jest kompilowany, a następnie dołączany do tworzonego programu. Jest to wygodna oraz bardzo popularna metodyka tworzenia rozwinięć języków programowania.

Wspomniane biblioteki nazywane są też bibliotekami serializacji danych. Ich typowym zastosowaniem jest komunikacja międzyprocesowa w sieciach komputerowych. za ich pomocą możemy w miarę bezproblemowo przesłać dane z programu napisanego w C++ na maszynie działającej pod kontrolą systemu Windows do drugiego programu, działającego na innej maszynie pod kontrolą systemu Linux, a napisanego np. w języku Python. W większości przypadków tego typu funkcjonalność wypełnia wymagania większości projektów.

Ograniczając się jedynie do języka C++, poświęcono jeszcze pewien czas na analizę produktu firmy CodeSynthesis – XSD. Jest to kompilator protokołów zapisanych w XML Schema do języka C++.

Oprócz wspomnianych rozwiązań można jeszcze wymienić wiele innych, np.: JSON, BSON, Action Message Format, Message Pack, oraz liczne mniej lub bardziej znane i związane z różnymi technologiami protokoły. Jednak w aspekcie przedstawionych wymagań na etapie projektowym uwagę przykuły jedynie następujące protokoły i narzędzia: Thrift, Protocol Buffers oraz parser XSD.

2. Komunikacja sieciowa w tworzonym systemie

Projekt sposobu komunikacji sieciowej pomiędzy systemem bazy danych a klientem był odkładany w czasie z uwagi na konieczność rozwiązania spraw podstawowych i bardziej istotnych dla konstrukcji tworzonego systemu strumieniowego. W pewnym momencie jednak wymóg przesłania i prezentacji danych stał się wymogiem funkcjonalnym, a ciągłe przesuwanie w czasie implementacji tej funkcjonalności tworzyło konieczność cyklicznej implementacji tymczasowych protez, które w dalszej perspektywie czasu musiały zostać zastąpione właściwym rozwiązaniem.

Jednym z takich wymagań funkcjonalnych jest stworzenie programu konsoli systemu strumieniowego. W przypadku tworzonego systemu zastosowanie typowej, tekstowej konsoli podobnej do konsoli systemu MySQL lub Oracle okazało się niewystarczające. W typowej konsoli systemu operacyjnego lub systemu bazy danych odpowiedź na wydane polecenie nie ulega późniejszym zmianom w czasie. System transakcyjny zapewnia fakt, że wysłanie zapytania do bazy danych daje odpowiedź spójną i jednoznaczną w danej chwili. W przypadku tworzonego systemu strumieniowego wysłane zapytanie do bazy danych rejestrowane jest w systemie, a odpowiedzi aktualizowane są na bieżąco wraz z napływem danych.

Podobne rozwiązania podglądu bieżącej zawartości strumieni danych znajdują się w systemach StreamBase [6] oraz Sybase CEP [7]. Wyżej wymienione rozwiązania przedstawiają strumienie danych wizualizowane za pomocą środowiska graficznego wewnątrz tabel. W tworzonym systemie z uwagi na przyszłe zastosowania założono istnienie jednego z najprostszych środowisk tekstowych – konsolę tekstową, w której odpowiedzi na zapytania będą aktualizowane na bieżąco. W końcowej fazie analizy projektowej przy uwzględnieniu wymagań, nie pozostało nic innego, jak napisać konsolę od podstaw na podstawie przenośnej biblioteki spełniającej wymagania funkcjonalne całego systemu.

2.1. Wybór biblioteki komunikacyjnej

Uwzględniając wymagania funkcjonalne opisane w punkcie 1.1 oraz biorąc pod uwagę przeprowadzoną analizę opisaną w punkcie 1.2, przeprowadzono wiele eksperymentów mających na celu określenie najbardziej efektywnego rozwiązania umożliwiającego implementację protokołu komunikacyjnego dla strumieniowego systemu zarządzania danymi.

Zadanie okazało się jednak szersze aniżeli funkcjonalność udostępniana przez opublikowane w literaturze standardowe rozwiązania. Obie biblioteki implementujące standardowe protokoły – tzn. Thrift oraz Protocol Buffers – oparto głównie na protokole z potwierdzeniami TCP. W tworzonym systemie jednak powstało wspominane w punkcie 1.1 zapotrzebowanie na dostarczenie danych protokołem strumieniowym. Typowo do tego celu używany jest protokół oparty na UDP. Wstępne próby rozszerzenia funkcjonalności analizowanych rozwiązań okazały się zbyt skomplikowane w krótkiej perspektywie czasowej.

W celu rozstrzygnięcia tego problemu uwagę skupiono uwagę na zupełnie innym rozwiązaniu – na bibliotece funkcjonującej od pewnego czasu w ramach zbioru bibliotek BOOST. Biblioteka ASIO [8] okazała się na tyle funkcjonalna, że umożliwiła stworzenie protokołu komunikacyjnego na podstawie TCP oraz UDP równocześnie.

Dodatkowo wewnątrz zbioru bibliotek BOOST znajduje się biblioteka Serialization implementująca protokoły binarne, JSON lub XML. W szczególności właściwe złożenie funkcjonalności biblioteki ASIO oraz Serialization może dać w rezultacie funkcjonalność biblioteki Thrift lub Protocol Buffers w aspekcie języka C++. Jednak przy projektowaniu protoku na potrzeby tworzonego systemu zarządzania danymi zastosowanie znalazła głównie biblioteka ASIO. Biblioteka Serialization była wykorzystana w innym aspekcie funkcjonowania systemu zarządzania danymi.

2.2. Konstrukcja protokołu komunikacji

System zarządzania danymi tworzony na podstawie wymagan czasu rzeczywistego na potrzeby systemów wbudowanych. Jedną z cech konstrukcyjnych tworzonego systemu jest unikanie zbędnych procesów opartych na wielowątkowości. Na chwilę obecną cały proces przetwarzania strumieni danych odbywa się w jednym wątku wraz z określonym z góry reżimem czasowym przeznaczonym na przetwarzanie kolejnych, ciągłych zapytań. Jeśli system nie jest w stanie spełnić reżimu czasu rzeczywistego i nie będzie w stanie przetworzyć zapytań w czasie założonym z góry, zgłosi on odpowiedni błąd i zaniecha przetwarzania danych.

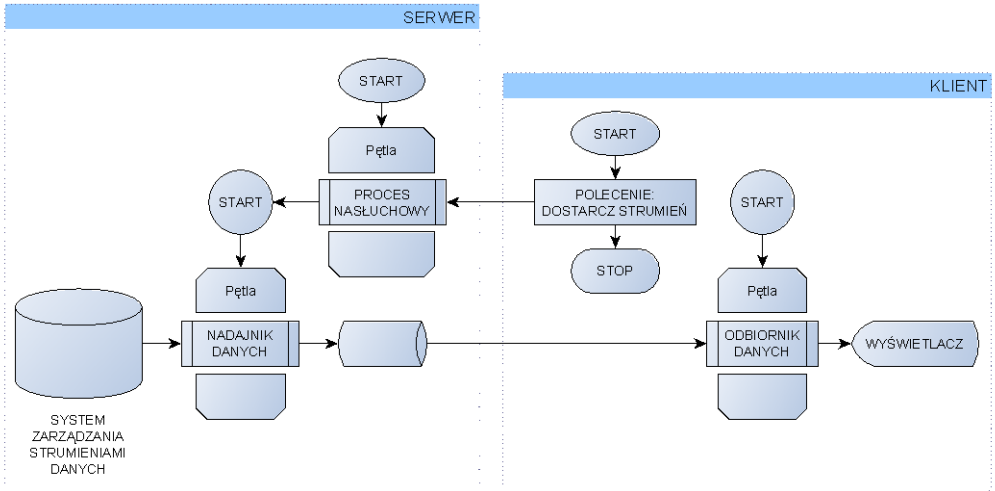
Przy implementowaniu modułu komunikacyjnego niestety nie udało się obecnie uniknąć wprowadzenia wielowątkowości w konstrukcji systemu. Na dzień dzisiejszy w systemie funkcjonują dwa podstawowe wątki. Pierwszy, związany z przewarzaniem strumieni danych, oraz drugi, odpowiedzialny za komunikację sieciową. Takie rozwiązanie podyktowane zosta-

ło analizą dostępnych opcji i sposobów komunikacji w innych systemach sieciowych, w szczególności w przykładach prostych serwerów sieciowych.

W wątku komunikacyjnym w domyślnym trybie pracy funkcjonuje proces pełniący funkcję nasłuchową, tzw. Listener. Proces nasłuchowy oczekuje połączeń na wybranym porcie komunikacyjnym. Protokół, który został użyty do implementacji tej usługi, oparto na protokole TCP [9]. Również tym protokołem system odpowiada ramką potwierdzenia informującą o sukcesie lub porażce otrzymanego żądania. Kolejnym działaniem, jakie podejmuje system, jest otwarcie portu obsługującego protokół strumieniowy UDP na porcie, którego numer jest losowany na bieżąco. Wylosowany numer zwracany jest również wraz z ramką potwierdzenia rejestracji zgłoszenia. Co ważne, protokół UDP może udostępniać tzw. funkcję multicast. Oznacza to, że strumienie danych udostępniane przez system bazy danych mogą być odbierane przez wielu klientów równocześnie. Z uwagi na fakt, że w systemie strumieniowym najważniejsza jest chwila obecna – tzn. to, co się dzieje tu i teraz – ta funkcjonalność protokołu UDP okazała się bardzo pożądana w tworzonym rozwiązaniu.

2.3. Obsługa błędów i wyznaczanie rozmiaru okna transmisji danych

Bezpośrednie zastosowanie istniejących rozwiązań ma jeszcze jedną wadę. Obsługa sytuacji nietypowych lub błędnych wymaga użycia określonego z góry przez twórców biblioteki scenariusza. W przypadku systemu strumieniowego należy rozważyć przypadki, w których dane nie dotrą do klienta lub zostaną przekłamane. Typowe rozwiązania komunikacyjne oparte na TCP tego typu funkcjonalność składają na barki protokołu. W przypadku protokołu UDP należy jednak takie sytuacje obsługiwać własnoręcznie. W rozważanym przypadku trzeba (powinno się) potraktować taką funkcjonalność protokołu bardziej jako zaletę aniżeli wadę.



Rys. 1. Schemat przepływu danych oraz sterowania w trakcie komunikacji
Fig. 1. Scheme of data and flow control during communication

Jedną z istotnych funkcji związanych z obsługą błędów jest procedura ustalania szerokości okna transmisji danych. W przypadku gdy dane nadawane są np. raz na sekundę, klient z reguły jest w stanie odebrać je z takim opóźnieniem. Problemy pojawiają się, kiedy dane nadawane są o wiele szybciej. W takiej sytuacji system, opierając się na tworzonych na bieżąco raportach z obsługi błędów, ustala wielkość okna transmitowanych danych. Procedura umożliwia ciągły i bieżący odbiór danych przez klienta z uwzględnieniem przesunięcia czasowego wymaganego na ich zbuforowanie i przesłanie. Informacja o przesunięciu czasowym jest dostarczana do klienta wraz z transmitowanym strumieniem danych.

3. Przykład użycia protokołu

Jak wspomniano na początku artykułu, w celu zapewnienia komunikacji pomiędzy systemem zarządzania danymi a klientem użyto biblioteki ASIO ze zbioru bibliotek BOOST. Bibliotekę ASIO zastosowano wewnątrz systemu zarządzania danymi, a także po stronie klienta, konsoli systemu zarządzania danymi.

Kilka sposobów użycia biblioteki zostało przedstawionych w kilkunastu przykładach dołączonych do dokumentacji biblioteki BOOST. W trakcie tworzenia własnego rozwiązania przedstawione przykłady stanowiły podstawę jego rozwoju. Opracowany i przyjęty przepływ danych i sterowania wewnątrz procesów klienta oraz serwera zobrazowano na rysunku 1.

3.1. Nawiązanie połączenia

Po stronie systemu zarządzania danymi na oddzielnym wątku funkcjonuje proces nasłuchowy, tzw. Listener. Nasłuch prowadzony jest na wybranym, stosunkowo niskim porcie za pomocą protokołu TCP. Arbitralnie wybrano wstępnie port do celów rozwojowych o numerze 1971.

W przypadku wspominanych w punkcie 1.2 bibliotek problem postaci i budowy ramki komunikacyjnej oraz szczegółów protokołu komunikacyjnego jest głęboko ukryty przed użytkownikiem. Podczas użycia biblioteki ASIO konieczne jest obsłużenie ramek samodzielnie. Wstępnie bardzo kusząca była propozycja oparcia protokołu komunikacyjnego z bazą danych na czysto binarnej postaci. Opracowanie prostego, binarnego protokołu wymiany danych i poleceń dla bazy danych nie stanowiło znaczącego problemu [1]. Jednak, mając na względzie poprzednie doświadczenia z poprawianiem protez i błędów projektowych, zdecydowano się na zastosowanie formatu XML jako formatu komunikacji w trakcie nawiązywania połączenia.

Wsparcie dla tego typu funkcjonalności ponownie odnaleziono wewnątrz biblioteki BOOST. Biblioteka pod nazwą Property Tree skrywa bardzo prostą i efektywną implementację

dostępu do danych zawartych wewnątrz formatu XML. Połączenie Property Tree wraz z ASIO w łatwy sposób rozwiązało problem przyszłych rozszerzeń protokołu komunikacyjnego.

Komunikacja z procesem nasłuchowym po stronie klienta odbywa się sekwencyjnie, bez konieczności użycia wątków. Klient tworzy zmienne typu property tree, zapytanie wypełnia poleceniem dla serwera, tworzy obiekt komunikacyjny ASIO po protokole TCP, wypełnia strumień poleceniem write_xml i wysyła dane do procesu nasłuchowego. Natychmiast przechodzi od oczekiwania na odpowiedź, odbierając z portu 1971 krótki komunikat o postaci formatu XML. Odpowiedź parsowana jest funkcją read_xml z biblioteki Property Tree.

Jeśli komunikat przewidywał udostępnienie danych strumieniowych, przez system bazy danych otwierany jest port po protokole UDP. Jeśli takie zapytanie już kiedyś wystosował do systemu inny klient i port został już uprzednio otwarty, kolejny klient otrzymuje ten sam numer portu i wtedy stosowany jest multicast.

3.2. Transmisja strumieni danych i prezentacja

Problemy występujące w trakcie transmisji strumieni danych w projektowanym systemie bardzo przypominają problemy związane z rozwojem serwisów internetowych typu YouTube. Podobnie jak w przypadku YouTube serwer dostarcza dane. Jednak zasadnicza różnica polega na tym, że w projektowanym systemie klientom dostarczane są te same dane w danej chwili. Szukając porównań do istniejących systemów, można by określić projektowany system jako YouTube transmitujący dane telewizyjne jedynie dla transmisji na żywo.

4. Wnioski i podsumowanie

Projektując i tworząc system komputerowy przeznaczony do przetwarzania i udostępniania danych, napotykamy wiele różnego rodzaju komplikacji. Zapis metody rozwiązywania i dokumentowania poszczególnych problemów badawczych i technicznych w drodze do osiągnięcia projektowanej funkcjonalności stanowi pewien dorobek umożliwiający podjęcie w przyszłości właściwych decyzji projektowych.

W artykule przedstawiono wybraną drogę oraz sposób rozwiązywania poszczególnych problemów napotkanych w trakcie tworzenia strumieniowego systemu zarządzania danymi. Surowej ocenie powinna podlegać poprawność wyboru poszczególnych decyzji projektowych. Przyjęta metodyka – polegająca na wstępnej analizie istniejących rozwiązań, przeglądzie dostępnych opcji i podjęciu działań mających na celu zdefiniowanie wymagań – stanowi przypadek wart opisania i przedstawienia szerszemu gronu.

Zdobyte doświadczenia wskazują na fakt, że dogłębne zrozumienie problemu oraz jego analiza w wielu przypadkach nadal są podstawowym wymogiem w trakcie tworzenia i projektowania nowych rozwiązań. Poruszanie się jedynie w sferze projektowej i analitycznej, przy założeniu za każdym razem istnienia gotowych do zastosowania w naszym przypadku rozwiązań, nie zawsze jest najlepszą drogą.

Wybrany sposób rozwiązania problemu stanowi kompromis pomiędzy użyciem istniejących i opracowanych bibliotek a stworzeniem wszystkiego od początku do końca. Jest to kompromis, którego wynikiem jest działająca i efektywna komunikacja pomiędzy klientem a serwerem w tworzonym systemie przetwarzania strumieni danych.

BIBLIOGRAFIA

1. Russel J., Cohn R.: Transparent Network Substrate. Book on Demand Ltd, 2012.
2. Widera M.: Integracja strumieniowego i relacyjnego systemu zarządzania danymi. Studia Informatica, Vol. 32, No. 2A (96), Gliwice 2011, s. 89÷102.
3. Upton E., Halfacree G.: Meet the Raspberry Pi. John Wiley & Sons, 2012.
4. Slee M., Agarwal A., Kwiatkowski M.: Thrift: Scalable Cross-Language Services Implementation. Facebook 2007.
5. Varda K.: Protocol Buffers: Google's Data Interchange Format. Google Technical Report, 2008.
6. StreamSQL Guide, StreamBase White Paper, 2010.
7. Heinze T.: Elastic complex event processing. Proceedings of the 8th Middleware Doctoral Symposium, Vol. 4, 2011, s. 1÷6.
8. Surhone L. M., Tennoe M. T., Henssonow S. F.: Asio C++ Library. VDM Publishing, 2010.
9. Ridge T., Norrish M., Sewell P.: TCP, UDP, and Sockets. Volume 3: The Service-level Specification. Technical Report, Computer Laboratory, University of Cambridge, 2009.

Wpłynęło do Redakcji 14 stycznia 2013 r.

Abstract

This paper present data stream management communication protocol. In first part of this paper there is a short review of existing communication and serialization libraries available on market. There is also short analysis of system requirements due to need of right review of

existing solutions. The target of developed solution are VPN, Desktop and Embedded systems [3]. In the first review author focuses on two libraries Thrift [4] and Protocol Buffers [5]. In final decision he uses BOOST ASIO library as basis for network transmission layer [8].

In the third part of this paper there is a use case presented. In Figure 1 – Scheme of data and flow control during communication there are almost all main data flow paths presented.

In use case part there are also presented detailed and specific data transmission details. Main handshake protocol is supported by TCP and stream transmission protocol is based on UDP. Almost all details of developed transmission protocol are supported by BOOST libraries. The TCP part of handshake protocol is successfully supported by XML property tree libraries.

As a results and summary part of this paper there is a successfully reached compromise between time and efficiency of developed solution.

Adres

Michał WIDERA: Streams Lab, ul. Reymonta 56/1, 41-800 Zabrze, Polska,
michal@streams.pl.