Uniwersytet Śląski
Wydział Matematyki, Fizyki i Chemii

Marcin Malich
Nr albumu: 211137

Praca licencjacka

Implementacja protokołu komunikacyjnego

Promotor:
dr Joachim Włodarz

Katowice, 2009

Wyrażam zgodę na udostępnienie mojej pracy dyplomowej dla celów naukowo-badawczych.

6 lipca 2009r. Marcin Malich

Słowa kluczowe: informatyka, sieci komputerowe, protokół, komunikacja.

Oświadczenie

Świadomy odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

6 lipca 2009r. Marcin Malich

Spis treści

1. Wprowadzenie

1.1. Wstęp

W obecnych czasach istotną rolę odgrywają sieci komputerowe, które od końca lat siedemdziesiątych zaczęły się rozrastać w zadziwiającym i niewyobrażalnym tempie. Zaczęto łączyć ze sobą małe, odizolowane od siebie sieci i stacje robocze, czego owocem jest dzisiejszy Internet - wielka globalna sieć komputerowa, dostępna dla wszystkich, z dowolnego miejsca na Ziemi.

Dzięki sieciom komputerowym stało się realne centralizowanie zasobów oraz przekazywanie danych z duża szybkością na duże odległości, co przekłada się na znaczący zysk nie tylko materialny. Dzięki takim zastosowaniom sieci komputerowe można spotkać wszędzie tam gdzie wykorzystywane są komputery, ułatwiając nie tylko pracę, ale także dając możliwość wspólnej rozrywki. Sieci komputerowe stały się nieodzowną i integralną częścią infrastruktury całego współczesnego informacyjnego świata i prawie każdej organizacji.

Szerokie stosowanie sieci komputerowych wymaga niezawodnych metod komunikacyjnych i transportowych. Technologia Ethernet jest obecnie najszerzej wykorzystywanym standardem do łączenia oraz wymiany danych miedzy komputerami pracującymi w sieciach lokalnych [1]. Istnieje wiele zdefiniowanych i ustandaryzowanych protokółów komunikacyjnych i transportowych wykorzystujących ramki ethernetowe [2].

Obecnie najpopularniejszymi i masowo wykorzystywanymi w sieciach protokołami transportowymi są TCP (*Transport Control Protocol*) [3] i UDP (*User Datagram Protocol*) [4]. Oba pracują w warstwie transportowej modelu OSI, bazując na tym samy protokole sieciowym - IP (*Internet Protocol*) [5]. Protokół TCP jest wykorzystywany aż w 90-ciu procentach sieci komputerowych, UDP w nieco mniejszym stopniu.

Protokoły te są przeważnie fundamentem dla innych protokołów pracujących w wyższych warstwach, głównie aplikacji. Na przykład protokoły HTTP, FTP, SSH wykorzystują tylko TCP jako niezawodny środek transportu danych, natomiast protokoły DNS, czy NTP posługują się transmisją datagramów dostarczaną przez UDP. Inne protokoły jak SMB używają obu protokołów transportowych.

Protokoły te idealnie sprawują się w globalnych i rozległych sieciach, ale czy na pewno

są również idealnym środkiem transportowym w dużo mniejszych i małych sieciach lokalnych, gdzie rządzą nieco inne prawa, a nacisk w transmisji kładzie się na inne aspekty?

Zdecydowanie nie są one idealnym rozwiązaniem, ich zalety w przypadku wielkich sieci globalnych stają się pewnego rodzaju wadami w lokalnych zastosowaniach. Można powiedzieć, że oba te protokoły plasują się na dwóch, skrajnie przeciwnych pozycjach.

TCP wydaje się zbyt ociężały, narzut danych oraz koszty nawiązania i podtrzymywania połączenia przez stos sieciowy mogą nie być akceptowane w specyficznych przypadkach. Jego główną zaletą w sieciach rozległych jest kontrola skuteczności transmisji, która wraz z mechanizmami retransmisji zapewnia gwarancję dostarczenia „przesyłki" do celu, a także transmisje pakietów we właściwej kolejności. W sieciach LAN mechanizmy te tracą na swojej wadze i stają się wręcz niepotrzebne, głownie ze względu na znikome ryzyko zaistnienia sytuacji, do przeciwdziałania których mechanizmy te powstały. Zatem zalety TCP mogą stać się nawet dosyć istotnymi problemami w zastosowaniach lokalnych.

Problemów takich nie posiada UDP, ale w jego przypadku sprawa wygląda zupełnie inaczej, można uznać, że protokół ten jest zbyt prosty. Dysponuje minimalnymi mechanizmami transmisji danych i opiera się bezpośrednio na protokole IP. Zatem wymaga dodatkowej abstrakcji, aby można było go prosto i szybko wykorzystywać, ale za to nadający się idealnie na fundament dla nowego protokołu.

Z powodu w/w aspektów i problemów istniejących protokółów, a także niemożności wykorzystania gotowych rozwiązań zdecydowałem się na próbę zaprojektowania bardziej dedykowanego do sieci lokalnych i pojedynczych segmentów sieci, protokołu komunikacyjnego. Protokołu bardzo prostego w implementacji, zwięzłego, nie generującego dodatkowych kosztów i narzutów w sferze sieciowej. Bezpiecznego pod względem poufności przesyłanych danych. Elastycznego medium transmisyjnego dającego się w bardzo prosty sposób rozszerzać i adaptować do wielu ciekawych zastosowań w sieciach lokalnych.

Najlepszym rozwiązaniem byłoby zaprojektowanie protokołu od podstaw, od zera, działającego na jak najniższych warstwach modelu sieciowego, ale jak się później okaże, w dalszej części niniejszej pracy, rozwiązanie takie nie jest, ani takie proste, ani współmierne do włożonego wysiłku. Dlatego warto czasem iść na kompromis i oprzeć się na solidnym fundamencie.

1.2. Cel pracy

Podstawowym celem niniejszej pracy jest zaprojektowanie protokołu komunikacyjnego działającego w sieci lokalnej, bądź pojedynczym segmencie sieci. Prostego i wydajnego, a zarazem bezpiecznego i elastycznego, dającego się w bardzo łatwy sposób rozszerzać. Bezpośrednim tego wynikiem jest specyfikacja protokołu SLCP.

Dodatkowym celem, obok projektu protokołu jest stworzenie przenośnej, programowej implementacji projektowanego protokołu w postaci biblioteki programistycznej, napisanej w języku C++. Umożliwiającej szybkie i proste wykorzystanie protokołu w rzeczywistych rozwiązaniach, takich jak komunikacja tekstowa miedzy użytkownikami sieci, czy zdalne sterowanie innymi urządzeniami i usługami dostępnymi w sieci lokalnej.

Obok implementacji biblioteki, celem zaprezentowania części możliwości protokołu, a także przetestowania samej biblioteki oraz zastosowanych rozwiązań, zostanie wykonany przykładowy program wykorzystujący stworzoną bibliotekę. Aplikacja oparta została o graficzny toolkit wxWidgets[1], zapewniający multiplatformowość i natywne GUI, przez co program prezentuje się bardzo ładnie oraz łatwo i intuicyjnie obsługuje. Przeznaczeniem tej aplikacji jest prowadzenie rozmów tekstowych pomiędzy użytkownikami sieci lokalnej.

1.3. Opis zawartości

Rozdział 1 jest krótkim wprowadzeniem do tematu problematyki i konieczności implementacji kolejnego protokołu komunikacyjnego, bardziej odpowiadającego aspektom i potrzebą sieci lokalnych. Przedstawia również niezbędne informacje i cele dotyczącej niniejszej pracy.

W **rozdziale 2** zaprezentowano ogólne cele i założenia projektowanego protokołu, a także powody i uzasadnienia dokonanych wyborów projektowych oraz wizje autora dotyczące idealnego protokołu transportowego przeznaczonego do sieci lokalnych.

Przykładowe scenariusze i przypadki użycia oraz wykorzystania projektowanego protokołu przedstawiono w **rozdziale 3**. Mam nadzieję, że jest to tylko wieszołek ogromnej góry lodowej ciekawych zastosowań.

Rozdział 4 jest typową, czysto techniczną specyfikacją protokołu SLCP, przedstawiającą budowę i formaty poszczególnych pakietów oraz ich przeznaczenie i obowiązki.

W kolejnym, **5-tym rozdziale** przedstawiono powstającą na równi z projektem protokołu SLCP obsługującą go bibliotekę programistyczną, napisaną w języku C++.

Praktyczny przykład wykorzystania stworzonej biblioteki i protokołu SLCP przedstawia **rozdział 6**. Prezentowany w nim program służy do prowadzenia rozmów tekstowych miedzy użytkownikami sieci lokalnej bez potrzeby uruchamiania jakiegokolwiek serwera.

Rozdział 7 zawiera krótkie podsumowanie przeprowadzonych prac i wnioski.

[1] W rzeczywistości nie jest to tylko „czysta" biblioteka do budowy GUI, zawiera wiele innych elementów, tworząc wspaniały framework dla cross-platformowych aplikacji.
Strona projektu: `http://wxwidgets.org`

1.4. Prawa autorskie

Implementacje biblioteki slcp i programu Lanek zawartych na płycie CD dołączonej do niniejszej pracy są objęte licencją GNU GPL (*GNU General Public License*) w wersji 2 i mogą być rozpowszechniane tylko i wyłącznie na zasadach określonych przez tą licencję. Pełna treść licencji znajduje się na stronie fundacji GNU[2], została także dołączona do w/w oprogramowania zawartego na CD-ROM-ie.

Autor niniejszej pracy nie ponosi żadnej odpowiedzialności za jakiekolwiek skutki i szkody wynikłe z wykorzystania przez użytkownika lub inną osobę materiałów i informacji zawartych w niniejszej pracy, uruchamiania oraz wykorzystania programów i bibliotek zamieszczonych na dołączonym do pracy CD-ROM-ie oraz ich nieprawidłowego działania. Każdy instaluje i uruchamia oprogramowanie wyłącznie na własną odpowiedzialność.

[2] `http://www.gnu.org/licenses/old-licenses/gpl-2.0.txt`

2. Projektowanie protokołu SLCP

Projekt protokołu SLCP został oparty na kilku głównych celach i założeniach, które odzwierciedlały wizje bezpiecznego i prostego protokołu komunikacyjnego dostosowanego do działania, w dużej mierze, w sieciach lokalnych i pojedynczych segmentach sieci.

W tym rozdziale zostaną szczegółowo omówione te kwestie i założenia jakie stawiano projektowanemu protokołowi oraz funkcje i możliwości jakie powinien spełniać. Nie zabraknie również uzasadnień dokonanych wyborów, przed jakimi, nieraz dosyć ważnymi i trudnymi stawał twórca, a które miały istotny wpływ na kształt samego protokołu.

Odnośnie nazwy protokołu, jego skrót może być rozwijany na różne sposoby, które trafnie mogą odzwierciedlać cel i przeznaczenie protokołu[1], aczkolwiek oficjalną nazwą i rozwinięciem skrótu jest *Secure LAN Communication Protocol*.

2.1. Geneza

Głównym celem powstania protokołu *Secure LAN Communication Protocol* było wypełnienie luki pomiędzy protokołami TCP i UDP, w głównej mierze, jak sama nazwa wskazuje, do działania w sieci lokalnej.

Jak zostało wspomniane w wprowadzeniu, w rozdziale 1, protokoły TCP i UDP niezbyt idealnie nadają się do niektórych zastosowań w sieciach lokalnych, gdzie w odróżnieniu od globalnych sieci oraz Internetu panują trochę inne warunki i pewne aspekty działania protokołów przewidzianych głownie do tych właśnie wielkich sieci mogą okazać się nieakceptowane z punktu widzenia sieci LAN, bądź pojedynczych segmentów sieci.

Zatem konieczne stało się opracowanie nowego protokołu dostarczającego prostych, praktycznych i bezpiecznych mechanizmów oraz metod komunikacji, kontroli i możliwości zdalnego wykonywania procedur pomiędzy dwoma hostami w sieci, grupą hostów, bądź hostem a wszystkim pozostałymi maszynami podłączonymi do sieci, na których uruchomiono daną usługę lub serwis.

[1] S można rozwinać do *simple* lub *secure*, a C do *control*, *command* czy *communication*, zatem jak widać możliwości rozwinięcia skrótu protokołu jest sporo.

2.2. Założenia i funkcje

Kilka założeń projektowych jakie przyświecały autorowi w czasie projektowania, oraz funkcji i możliwości jakie powinien udostępniać, dzięki którym stał się użytecznym narzędziem i łatwym do wykorzystania w najróżniejszych celach.

2.2.1. Elastyczność i prostota

Jednym z nadrzędnych założeń i celów projektu było stworzenie prostego i elastycznego protokołu. Pozwalającego na wykorzystanie jego możliwości w różnych rozwiązaniach użytkowych, łatwość adaptacji w różnych środowiskach oraz możliwość prostej implementacji programowej.

Obok tego starano się stworzyć protokół, którego rozszerzalność będzie bardzo prostym zabiegiem. A przykładowa implementacja protokołu w postaci biblioteki programistycznej, również powinna posiadać takie cechy.

Prostota rozszerzalności i wykorzystania jest istotnym kryterium przy wybieraniu bazowego budulca dla nowego protokołu pracującego w wyższych warstwach sieciowych i abstrakcji. Zamierzeniem autora było stworzenie właśnie takiego bazowego protokołu, nie tylko dla bardziej specyficznych protokółów aplikacyjnych i zastosowań, ale także dla ciekawych, praktycznych usług i aplikacji.

2.2.2. Komunikacja

Komunikacja jest nadrzędną funkcją projektowanego protokołu, więc chyba nie trzeba dużo o niej mówić. Jedynie warto zwrócić uwagę, że wymiana informacji, obok zapewnionego bezpieczeństwa, powinna łatwo dawać się zaadaptować do różnych środowisk.

Zatem metody wymiany informacji i danych zależne będą w dużym stopniu od konkretnego celu w jakim protokół zostanie wykorzystany. A co za tym idzie, ścisła specyfikacją formatu i zachowania tych metod nie jest możliwa tak, aby nie naruszyć poprzednich założeń związanych z elastycznością.

2.2.3. Transmisja grupowa

Kolejnym założeniem jest łatwa i wydajna dystrybucja informacji do pojedynczego, grupy, bądź wszystkich odbiorców w sieci. Efektywna komunikacja z grupą hostów, daje bardzo szerokie pole działania. Dzięki transmisji grupowej będzie można uniknąć wielokrotnego wysyłania tych samych danych do poszczególnych hostów. Transmisja taka możliwa jest przez wykorzystanie techniki *brodcast* i *multicast*.

Korzystając z rozgłoszeniowego trybu transmisji, czyli techniki brodcast, docelową grupą odbiorców są wszystkie hosty podłączone do sieci, domeny broadcastowej [6].

W przypadku techniki multicast, do grupy odbiorców można przypisywać dowolne hosty z sieci, dzięki czemu grupą odbiorczą może być ściśle określona liczba hostów [7]. Zarządzanie grupami odbiorców i przynależnością możliwe jest przez wykorzystanie odpowiedniego protokołu - IGMP (*Internet Group Management Protocol*), za pomocą którego można powiadamiać routery w sieci o chęci przyłączenia się do lub odejścia z określonej grupy multicastowej [8].

Korzystając z obu tych technik, można przyjąć, że istnieje adres grupowy, który zapewni efektywną i prostą komunikację z całą grupą. Dodatkowo poprzez adresowanie grupowe, które może identyfikować nie tylko grupy maszyn, ale wszystkie hosty danego segmentu lub sieci, bardzo proste staje się zbiorowe kontrolowanie i zarządzanie uruchomionymi usługami na poszczególnych hostach. A także, w niektórych specyficznych usługach, bazujących na projektowanym protokole, adres grupowy, może być traktowany jako „wirtualny" serwer, co zapewnia skuteczną komunikację pomiędzy wszystkimi zainteresowanymi hostami.

Sam protokół nie powinien ściśle definiować techniki jaka zostanie wykorzystana do grupowej transmisji, kwestia ta leży w gestii implementacji programowej, specyfikacja zakłada jedynie istnienie takiej możliwości i transparentne z niej korzystanie.

2.2.4. Identyfikacja punktów końcowych

Obok możliwości prowadzenia grupowej dystrybucji datagramów, istotnym założeniem jest również możliwość uruchomienia na jednej maszynie kilku aplikacji, usług czy serwisów korzystających z protokołu SLCP. Aby spełnić to założenie wymagany jest odpowiedni mechanizm identyfikowania procesów działających na danym hoście.

Mechanizm identyfikacji punktów końcowych w przypadku protokołów warstwy transportowej, takich jak TCP i UDP, oparty jest na portach. Konkretny port jest identyfikowany za pomocą 16-bitowej liczby (0 - 65535). Każdy pakiet w/w protokółów zawiera w swoim nagłówku pola określające port nadawcy i odbiorcy pakietu.

Powiązanie procesu z danym portem, zwane bindowaniem, umożliwia nasłuchiwanie i odbieranie tylko tych pakietów, które jako miejsce docelowe zawierają ten ściśle określony numer portu oraz wysyłanie pakietów, których źródłem jest również ten ustawiony port. Dzięki numerom portów można jednoznacznie określić w systemie proces do lub z jakiego dane powinny trafić lub zostały wysłane, co nieraz pozwala na identyfikację usługi lub aplikacji uruchomionej na odległej maszynie zdalnej.

Oczywiście różne usługi mogą używać tych samych numerów portu, ale wtedy muszą korzystać z innego protokołu transportowego. Istnieją usługi korzystające jednocześnie

z jednego numeru portu i obu protokołów[2], a także wykorzystujące dwa różne porty do różnych zadań[3].

W zasadzie numery portów można podzielić na trzy grupy. Są to „dobrze znane numery portów" (ang. *Well Known Ports*) (0 - 1023) zarezerwowane na standardowo przypisane do nich usługi, zarejestrowane porty (1024 - 49151) i dynamicznie przydzielane podczas połączenia wraz z prywatnymi portami. Listę wszystkich zarejestrowanych portów można znaleźć na stronie internetowej IANA[4].

2.2.5. Zdalny nadzór i wywoływanie procedur

Funkcje pozwalające na kontrolę i nadzór zdalny mogą okazać się bardzo przydatne w zastosowaniach związanych z nadzorowaniem i kontrolą pewnych usług uruchamianych w sieci. Podobnie z możliwością wywoływania zdalnie procedur.

Mechanizmy takie mogą bardzo zwiększyć możliwości i cele w jakim można wykorzystać protokół, szczególnie w połączeniu z transmisją grupową, nabierają zupełnie innego wymiaru, dając nieocenione możliwości. Zatem istotnym celem jest zaimplementowanie w miarę prostych metod ułatwiających wykorzystanie protokołu SLCP w tych obszarach.

2.2.6. Bezpieczeństwo

Istotną kwestią na jaką należy zwrócić baczną uwagę jest bezpieczeństwo. Bezpieczeństwo w systemach informatycznych to nie tylko zapewnienie nienaruszalności i poufności danych przed niepowołanym dostępem, czy modyfikacją, to także dostępność, użyteczność oraz niezawodność [9].

Dane przesyłane siecią komputerową narażone są na różnego rodzaju niebezpieczeństwa, do których można zaliczyć zakłócenia przepływu informacji, przechwycenie danych, czy ich modyfikacja przez niepowołaną osobę, lub podszywanie i generowanie fałszywych informacji.

Podstawowym aspektem, z punktu widzenia projektowanego protokołu, jest bezpieczeństwo komunikacji, polegające na zapewnieniu poprawności przesyłania danych oraz ich integralności i zabezpieczenia przed próba podsłuchania, modyfikacji, czy zniszczenia podczas transmisji. Protokół powinien wspierać możliwość odpowiedniego zabezpieczenia przesyłanych danych, w głównej mierze przed niepowołanym dostępem oraz modyfikacją. Dlatego szyfrowanie oraz mechanizmy pozwalające na stwierdzenie integralności przesyłanych danych, powinny zostać zaimplementowane.

[2]Przykładem może być usługa DNS korzystająca z poru 53 za pomocą TCP i UDP.

[3]Protokół FTP używa portu 21 do poleceń i zależnie od rodzaju połączenia, 20 lub portu powyżej 1024 do przesyłania danych.

[4]`http://www.iana.org/assignments/port-numbers`

Przy projektowaniu protokołu należy także wziąć pod uwagę problem skalowalności, jakakolwiek zmiana, rozszerzenie struktur sieci, nie powinno wpływać na system, ani zakłócać bezpieczeństwa, komunikacji, czy integralności przesyłanych danych.

Osobną kwestią, związaną z bezpieczeństwem, jest problem zapewnienia odpowiedniej jakości usług (ang. *Quality of Service*). QoS jest zbiorem mechanizmów, które powinny zapewnić określone parametry transmisji, w celu osiągnięcia odpowiedniego poziomu jakości danej usługi, dającego satysfakcje końcowemu użytkownikowi [10].

Z punktu widzenia sieci lokalnych i pojedynczych segmentów sieci, problem z zapewnieniem odpowiedniej jakości nie powinien mieć wielkiego znaczenia. W sieciach LAN trudno o zaburzenie transmisji danych, czy utratę części przesyłanych pakietów, aczkolwiek jest to znaczące i realne zagrożenie[5].

2.3. Rozwiązania

Rozwiązania jakie zostały wybrane podczas całego procesu projektowego oraz analizy stawianych założeń i celów mogą wymagać wyjaśnień. Dlatego warto przedstawić omówienie i argumentację umożliwiając głębsze poznanie procesu wyboru odpowiednich środków, uznanych przez autora, za trafne i odpowiadające stawianym wymaganiom.

2.3.1. Protokół transportowy

Analizując wymagania i założenia projektowe opisane we wcześniejszych podrozdziałach, jako idealny fundament i protokół transportowy dla protokołu SLCP wybrano UDP.

Przeznaczeniem protokołu SLCP są sieci lokalne, więc może nasunąć się zasadnicze pytanie, dlaczego jako podstawę transportową został wybrany akurat protokół UDP, a nie próbowano zejść kilka warstw niżej i uplasować SLCP na warstwie sieciowej lub transportowej?

To prawda, można było wykorzystać adresy sprzętowe w komunikacji i pozbyć się narzutu oraz kosztów wynikających z użycia dodatkowych protokołów warstw wyższych - sieciowej (IP) i transportowej (UDP), ale pociągałoby to za sobą dodatkowe ograniczenia i multum nowych problemów. Oczywiście wszystko to zależy od przyjętych celów jakie chce się spełnić w projektowanym protokole.

Jednym z głównych założeń protokołu, obok prostoty była możliwość grupowej transmisji danych oraz uruchomiania na jednej maszynie kilku aplikacji, usług czy serwisów bazujących na protokole SLCP. Problemem grupowej transmisji w warstwach niższych nie

[5]Bezpośrednią przyczyną takich problemów może być zbyt nadmierne obciążenie sieci, co może powodować znaczne opóźnienia, skutkujące zaburzeniem kolejności przesyłanych pakietów, bądź ich utratę i nieosiągnięcie wyznaczonego celu.

istnieje, ponieważ nawet adresowanie MAC posiada odpowiedni adres brodcast, a technika multicast potrafi także korzystać z tego typu adresowania.

Jedyny istotnym tutaj problemem jest wymagany mechanizm identyfikacji punktów końcowych, co w sytuacji „zejścia" poniżej warstwy transportowej wymagałoby implementacji takiego mechanizmu ręcznie, co mogłoby się okazać nie tak całkiem prostą rzeczą, a także wymagającą specjalnych zabiegów i przywilejów[6].

W celu uniknięcia w/w problemów zdecydowano się na wykorzystanie istniejącego protokołu transportowego. Obecnie najpopularniejszymi i masowo wykorzystywanymi w sieci protokołami służącymi do transportu danych są TCP i UDP. Oba wykorzystują ten sam protokół sieciowy, czyli IP, a także używają portów do identyfikacji punktów końcowych.

Transmisja grupowa bazująca na technice multicast, nie jest mechanizmem zorientowanym na połączenia, dlatego protokoły takie jak TCP są nie odpowiednie. Ponadto TCP jest zbyt wielki i ociężały, aby elastycznie wykorzystywać go w komunikacji lokalnej. Narzut danych i większa liczba przesyłanych pakietów, a także koszty nawiązania połączenia i podtrzymywania sesji przez stos sieciowy dyskredytują go w naszym zastosowaniu. W przypadku UDP sprawa wygląda zupełnie odwrotnie, nie ma narzutu na nawiązywanie polaczenia i podtrzymywania sesji, jest prosty i elastyczny, więc idealnie nadaje się na fundament naszego protokołu.

Transmisja poprzez UDP w porównaniu do TCP daje znaczące zyski ilości wysyłanych danych, ale mimo to nadal przy małych porcjach danych, generowane nadwyżki mogą być duże, w większości przez nagłówek datagramu IP. Dla porcji danych mieszczących się w jednej ramce ethernetowej, ilość dodatkowych informacji wyniesie około 46 bajtów, łącznie z nagłówkiem UDP. Na większość tych problemów nic nie poradzimy, choć istnieje kilka protokółów zajmujących się zmniejszeniem nadwyżek, chociażby poprzez kompresje nagłówków, więcej informacji można znaleźć w dokumentach RFC [11][12].

Istotną kwestią jest również długość przesyłanych danych przez protokół SLCP, sam protokół nie definiuje żadnych limitów co do długości danych. Jedynym ograniczeniem jest maksymalna możliwa porcja danych jaka może przenieść datagram UDP, która teoretycznie wynosi 65,527 bajtów. Wartość ta pomniejszona o długość nagłówka SLCP jest maksymalna długością danych jaka może być przesłana przez pakiet SLCP.

Odnośnie kwestii bezpieczeństwa, dalsze podrozdziały poruszają ten temat, tutaj jedynie należy wspomnieć, że sam protokół UDP nie dostarcza żadnych mechanizmów służących do kontroli przepływu danych, ani gwarancji jakości usług. Kwestie tą poruszaliśmy przy opisie założeń związanych z bezpieczeństwem[7].

[6]Gniazda typu SOCK_RAW i gniazda domeny PF_PACKET na systemach uniksopodobnych mogą otwierać tylko procesy z EUID=0 lub posiadające włączony atrybut CAP_NET_RAW.

[7]Patrz podrozdział 2.2.6.

2.3.2. Szyfrowanie danych

Problem bezpieczeństwa transmisji danych został rozwiązany przez zastosowanie w protokole SLCP szyfrowania danych przy użyciu asymetrycznego algorytmu RSA. Algorytm ten przy zastosowaniu rozsądnych rozmiarów kluczy zapewnia wysokie bezpieczeństwo.

Wybór algorytmu asymetrycznego był uwarunkowany kilkoma powodami i zasadą działania protokołu SLCP.

Kryptografia asymetryczna bazuje na zestawie dwu lub więcej powiązanych ze sobą kluczy, umożliwiających wykonywanie pewnych czynności kryptograficznych. W przypadku szyfrowania wykorzystuje się dwa klucze: publiczny oraz prywatny. Publiczny służy do szyfrowania informacji i jest udostępniany każdemu kto chce zaszyfrować wiadomość. Klucz prywatny znajduje się w wyłącznym posiadaniu adresata i tylko on korzystając z niego może odczytać zaszyfrowane wiadomości.

Kryptografia asymetryczna korzysta z algorytmów wykorzystujących operacje jednokierunkowe, czyli takie które łatwo można przeprowadzić w jedną stronę, ale trudno w drugą, co objawia się tym, że korzystanie z tego typu algorytmów jest dużo kosztowniejsze niż wykorzystanie innego rodzaju kryptografii[8].

Dużo „lżejsze" są algorytmy symetryczne, w których używa się tego samego klucza do szyfrowania i odszyfrowania wiadomości, przez co bezpieczeństwo jest również mniejsze, w dużej mierze zależne od długości klucza.

W praktyce prawie nigdy nie szyfruje się wiadomości za pomocą algorytmów asymetrycznych. A zamiast tego korzysta się z połączenia kilku tych technik w celu osiągniecia zadowalających wyników i bezpieczeństwa względem użytych zasobów i kosztowności samego procesu kryptograficznego.

Praktykowana metoda jest szyfrowanie algorytmem asymetrycznym tylko samego klucza jakiegoś algorytmu szyfrowania symetrycznego, np. szyfrowanie blokowe algorytmem AES, co zapewnia bezpieczną wymianę newralgicznego elementu tej techniki kryptograficznej. Dalsze szyfrowanie przebiega wykorzystując dużo szybszy algorytm symetryczny.

Technika taka niestety nie sprawdziłaby się w naszym protokole zbyt dobrze, problemem byłaby nadmiernie rozbudowana metoda uzgadniania kluczy szyfru symetrycznego. A przekazanie samego klucza w pakiecie inicjującym lub korzystanie z algorytmu symetrycznego w całej komunikacji równoważne byłoby zerowym bezpieczeństwem.

Dużo lepszym rozwiązaniem jest wykorzystanie jednej z trybów stosowanych w szyfrach blokowych[9].

[8] Nie oznacza to jednoznacznie, że operacje jedno kierunkowe są kosztowniejsze od innych, ale przeważnie w algorytmach asymetrycznych trzeba wykonać dużo więcej obliczeń i działań, co rzutuje na wynik złożoności i narzutu.

[9] `http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation`

Najszerzej stosowanym trybem jest CBC (*Cipher Block Chaining*) - tryb wiązania bloków zaszyfrowanych. Jego działanie jest bardzo proste, przed zaszyfrowaniem każdy z bloków wiadomości jest przekształcany funkcją XOR z szyfrogramem uzyskanym z szyfrowania poprzedniego bloku. Szyfrowanie pierwszego bloku się trochę różni, ponieważ nie istnieje szyfrogram poprzedniego bloku, więc stosuje się tzw. wektor początkowy IV (*initial vector*), składający się z wylosowanego ciągu bitów, który jest dołączany do wiadomości, bądź wymieniany razem z kluczem.

Tryb CBC sprawdza się bardzo dobrze w sytuacjach, gdy istnieje możliwość powtórzenia się bloków szyfru. Przy stosowaniu CBC, gdy zaszyfrujemy dwa identyczne bloki tekstu jawnego, otrzymamy dwa różne szyfrogramy, co skutecznie eliminuje wszelkie ataki i analizy.

Kryterium wykluczającym tryb CBC z naszego zastosowania jest uzależnienie się od kolejności wysyłanych i odbieranych pakietów, co w naszym protokole nie zawsze musi być spełnione, zależnie od wykorzystania. Zatem w przypadku naruszenia kolejności w transmisji danych lub innych błędów komunikacyjnych niemożliwe będzie poprawne rozszyfrowanie danych.

Do argumentów wykluczających użycie symetrycznego algorytmu, obok rozbudowanej negocjacji i wymiany kluczy, należy jeszcze dodać wymagane użycie innego klucza dla każdego połączenia *host-to-host*, w celu wyeliminowania możliwości podsłuchu komunikacji. Częściowo ten problem mógłby rozwiązać tryb CBC z różną wartością wektora inicjującego dla każdego hosta. Niemniej użycie algorytmu asymetrycznego z dwoma kluczami zwalnia z rozbudowanego sposobu rozpowszechniania klucza, wystarczy wszystkim udostępnić ten sam klucz publiczny nie martwiąc się o niepowołany dostęp do przesyłanych danych.

Po analizie przedstawionych argumentów i technik kryptograficznych jakie rozważano zastosować w protokole SLCP, tak jak napisano na początku niniejszego podrozdziału, zdecydowano się na wykorzystanie algorytmu RSA z kluczem 1024-bitowym. Mimo, iż wybór ten pociąga za sobą nieco większe wykorzystanie zasobów hosta, mam nadzieję, że wysoki poziom bezpieczeństwa oraz inne ułatwienia rekompensują wszelkie niuanse związane z wykorzystaniem tego algorytmu asymetrycznego.

Szyfrowane pakiety oraz wysyłane do grupy, bądź wszystkich hostów są odpowiednio oznakowane w nagłówku, a co za tym idzie, komunikacja w obrębie grupy również jest zabezpieczona, jeśli tylko istnieje taka potrzeba. Wystarczy dla wiadomości „wspólnych" używać tego samego zestawu kluczy. Protokół nie specyfikuje żadnego sposobu wymiany kluczy grupowych, ale do tego celu można użyć odpowiednich narzędzi oferowanych przez protokół i zaimplementować taki mechanizm, jeśli będzie on rzeczywiście wymagany w docelowym zastosowaniu.

2.3.3. Mechanizmy integralności

Kontynuując wątek związany z bezpieczeństwem, protokół SLCP nie posiada żadnych mechanizmów służących do stwierdzania naruszenia integralności danych i całego pakietu. Możemy opierać się jedynie na takich mechanizmach implementowanych w używanych protokołach warstw niższych, na których bazuje sam protokół SLCP, czyli IP i UDP.

Protokół IP dba jedynie o swój nagłówek, 16-bitowa suma kontrolna nagłówka, zapewnia integralność informacji zawartych tylko w samym nagłówku datagramu IP. Wartość tej liczby liczona jest według prostego algorytmu, który najpierw sumuje wszystkie słowa nagłówka, a następnie przeprowadza negacje bitową otrzymanej liczby oraz konwersuje do szerokości 16 bitów [5].

Nieco lepiej wygląda sprawa w UDP, pole sumy kontrolnej zawarte w datagramie UDP obliczane jest z nagłówka i danych. Algorytm jest nieco bardziej złożony niż w przypadku IP, jego opis można znaleźć w odpowiednim dokumencie RFC protokołu UDP [4].

Sprawdzenie sumy kontrolnej pozwala upewnić się, że integralność pakietów nie została naruszona. W przypadku naruszenia datagramy takie zostają odrzucone już na poziomie odpowiedniej warstwy sieciowej, na której pracują i żadna informacja nie zostaje wyżej przekazana. Może to prowadzić do różnych problemów w komunikacji, ponieważ protokół SLCP nie implementuje żadnych funkcji retransmisyjnych.

Powyższe rozważania odnośnie protokołu IP dotyczyły implementacji IP w wersji 4, który nie posiada mechanizmów służących do sprawdzania integralności i zapewnienia poufności przesyłanych danych. Problem ten można rozwiązać przez używanie protokołu IPsec.

Głównymi zadaniami IPsec jest ochrona integralności i poufność danych, przy czym IPsec posiada istotną cechę - przezroczystość dla protokołów warstw wyższych. Protokół IPSec dostarcza dwóch protokołów do zapewnienia bezpieczeństwa transmisji, są nimi AH (*Authentication Header*) oraz ESP (*Encapsulation Security Payload*).

Protokół AH odpowiada za ochronę integralności danych i nagłówka pakietu, używając metod takich jak MD5, SHA1, RIPEMD-160 [13]. Protokół ESP odpowiada za poufność przesyłanych danych wykorzystując algorytmy DES, 3DES, Blowfish, AES [14].

IPv6 traktuje IPsec jako integralną cześć protokołu, przez dodanie nagłówków, służących właśnie do celów wykorzystywanych przez AH i ESP.

2.3.4. Model zdalnej kontroli i wywoływania procedur

Do zdalnej kontroli można wykorzystać mechanizmy wywołania procedur, gdzie na wysłane zadanie wykonania polecenia otrzymamy odpowiednią odpowiedź zwrotną zwierającą wynik. W ten sposób można kontrolować w prosty sposób jakiekolwiek usługi i urządzenia

zaopatrzone w odpowiednie oprogramowanie.

Nie jest to jedyny sposób nadzoru, czy kontroli. W tym celu można wykorzystać także inne mechanizmy jakie posiada protokół. Mowa tutaj o pakiecie statusu wysyłanym przez hosta w odpowiedzi na zaobserwowanie jakiegoś zdarzenia lub wynikniecie sytuacji, o której należy poinformować. Prócz tego protokół SLCP posiada bardzo prosty środek do kontroli aktywnej usługi, szeroko wykorzystywany w sieci mechanizm powiadamiania co określony odcinek czasu, że żyjemy - ping.

Przechodząc do kwestii zdalnego wywoływania procedur, godna polecenia jako doskonałe wprowadzenie do tematyki, a także dobre źródło służące do poznania i zrozumienia idei i koncepcji związanych ze zdalnym wywoływaniem procedur jest publikacja Birrella i Nelsona [15].

Model wywoływania zdalnie procedur w protokole SLCP opiera się na jednym pakiecie, który wykorzystywany jest w roli zapytania i odpowiedzi na nie udzielonej, zależnie od kontekstu. Model taki odpowiada rzeczywistości, na takiej zasadzie oparte są wszystkie dedykowane do tego celu protokoły jak RPC [16].

Istotnym elementem jest możliwość przekazywania argumentów, w przypadku SLCP zdecydowano się na przekazywanie danych w postaci ciągu danych binarnych jako jedynego parametru. Najlepszym rozwiązaniem jest traktowanie tego ciągu jak pewnej struktury uporządkowanych danych, co może odpowiadać strukturom danych języka C.

Taki model działania ma dwie istotne zalety, możliwość przekazania w strukturze dowolnych danych i bardzo prostą implementację. Nie musimy się martwic o możliwość przekazywania nieznanej ilości parametrów. Odpowiednią interpretacją danych powinna zając się sama aplikacja.

Kolejnym aspektem jest istota działania, w większości protokołów, z punktu widzenia użytkownika, wywołanie procedury zdalnej jest podobne do wywołania lokalnego, w zasadzie niczym się nie różni, po wywoływaniu procedury następuje wysłanie zapytania i oczekiwanie na odpowiedź, blokując przez ten czas dalsze przetwarzanie programu.

W przypadku SLCP, protokół nie narzuca takiego działania, decyzja o blokowaniu lub nie działania na czas wykonywania procedury powinna być uwarunkowana i zależna od konkretnego zastosowania. Zatem protokół nie ingeruje w to zachowanie, dając wolną rękę.

Identyfikacja i powiązanie zapytań z odpowiedziami jest możliwe za pomocą odpowiedniego numeru identyfikującego, rozwiązanie takie daje nieco większe pole manewru.

3. Scenariusze użycia

W rozdziale tym opisano niektóre aspekty zastosowania protokołu SLCP. Przykłady te zostały wybrane w celu zilustrowania podstawowych operacji i możliwości aplikacji, czy usług bazujących na SLCP. Nie są to wyłączne dziedziny i zastosowania w jakich można wykorzystać ten protokół. Mam nadzieje, że istnieje wiele innych ciekawych zastosowań protokołu SLCP, a prezentowane przykłady są tylko czubkiem ogromnej góry lodowej.

3.1. Prosta komunikacja tekstowa

Wszelkie komunikatory i inne programy do rozmów w czasie rzeczywistym są bardzo popularne nie tylko w Internecie, ale także w sieciach lokalnych. Choć ostatnio można zauważyć mały zanik tych drugich i wyparcie ich przez zwykle internetowe komunikatory.

Historia komunikacji tekstowej narodziła się w latach 70-tych, za sprawą powstania komendy `talk`, która umożliwiała bezpośrednią rozmowę z innymi użytkownikami systemu. Pierwsza wersja tego programu pojawiła się na systemie komputerowym DEC PDP-11. W 1983 roku aplikacja ta pojawiła się na systemach unikowych (BSD v4.2), wzbogacona o protokół umożliwiający komunikację miedzy użytkownikami dowolnych maszyn w sieci[1].

Punktem zwrotnym, a także początkiem ery komunikatorów internetowych było wydanie w 1996 roku przez izraelską firmę Mirabilis programu ICQ (*I Seek You*), który zdominował na długo rynek komunikatorów i stal się wyznacznikiem oraz wzorem. Dziennie w sieci ICQ przesyłane jest ok. 400 milionów wiadomości.

Kolejnym przełomowym wydarzeniem było rozpoczęcie w 1998 roku projektu Jabber przez Jeremie Millera, którego celem było powstanie otwartych i zdecentralizowanych rozwiązań komunikacji natychmiastowej, bazujących na tekstowym protokole opartym na języku XML. Jabber stał się podwaliną do powstania standaryzowanego w 2004 roku protokołu XMPP (*eXtensible Messaging and Presence Protocol*), który wykorzystuje wiele rozwiązań, wykorzystanych w projekcie Jabber[2].

[1] `http://en.wikipedia.org/wiki/Talk_(Unix)`

[2] XMPP jest jakby nowszą wersją Jabbera, mimo, iż XMPP zakłada kompatybilność wstecz, bardzo stare serwery Jabbera nie są w pełni zgodne z XMPP.

Od tego czasu powstały coraz to nowsze rozwiązania wspomagające komunikację tekstową, a także głosową i wideokonferencje.

Protokół SLCP nadaje się idealnie do zastosowania w komunikacji natychmiastowej działającej w sieci lokalnej. Zastosowanie to było jednym z pierwszych powodów dla których autor postanowił przyjrzeć się bliżej możliwością komunikacji w sieciach LAN oraz było jedną z pierwszych przyczyn, które doprowadziły do narodzin protokołu SLCP.

Oczywiście najlepszym rozwiązaniem do komunikacji natychmiastowej jest wykorzystanie przeznaczonych do tego celów środków, takich jak XMPP. Sprawa komplikuje się w sytuacjach, gdy nie ma możliwości ani warunków, nie tylko technicznych do tego, aby uruchomić dodatkową maszynę w sieci, w roli serwera.

W takich sytuacjach rozwiązaniem może być wykorzystanie protokołu SLCP, który posiada odpowiednie mechanizmy zapewniające bezproblemową i bezpieczną komunikację. Bazując na tym protokole można stworzyć idealne rozwiązanie spełniające postawione zadania komunikacji natychmiastowej bez potrzeby uruchamiania dodatkowych narzędzi, czy serwerów. Komunikacja odbywa się bezpośrednio pomiędzy uruchomionymi aplikacjami w sieci, a rolę „wirtualnego serwera" spełniają mechanizmy grupowej transmisji danych, jak brodcast i multicast.

Mechanizmy protokółu nadzorujące połączenie, w postaci pakietów `INIT`, `BYE` i `PING` zapewniają kontrolę połączeń danego klienta z siecią oraz powiadamianie o zmianach i nowych podłączeniach w sieci.

Pakiety typu `STATUS` oraz `MESSAGE` wykorzystujemy zgodnie z ich przeznaczeniem. Informacje o zmianach statusu wraz z zawartym opisem stanu są idealnym odzwierciedleniem informowania o dostępności w dedykowanych rozwiązaniach IM. Wiadomości transportowane pakietem `MESSAGE` stają się podstawowym środkiem komunikacji tekstowej.

Pakiet `COMMAND` jest bardzo dobrym fundamentem do implementacji dodatkowych możliwości i funkcji, jakie chcemy zaoferować naszym użytkownikom. Pobieranie informacji o użytkowniku (whois), wykorzystanie do zestawiania połączeń konferencyjnych, lub transferu plików miedzy użytkownikami, są tylko garstką niezliczonych zastosowań i możliwości jakie można stworzyć.

Dzięki szyfrowaniu danych dostępnym w protokole SLCP, wszelka komunikacja oraz wymiana wiadomości jest bardzo bezpieczna. Nie musimy się martwic o jakiekolwiek podsłuchanie w sieci przesyłanych danych.

Realnym rozwiązaniem bazującym na protokole SLCP, umożliwiającym bezpośrednią wymianę wiadomości między użytkownikami sieci lokalnej jest program Lanek, opisany w rozdziale 6. Jest on przykładową, prostą implementacją, bazującą na bibliotece slcp, opisanego tutaj scenariusza wykorzystania protokołu SLCP.

3.2. Zdalna kontrola i zarządzanie

Dzięki mechanizmom zdalnej kontroli i wywoływaniu procedur protokół SLCP idealnie nadaje się do kontrolowania i zarządzania różnymi usługami w sieci.

Przykładem tutaj może być niestandardowe zarządzanie sprzętem infrastruktury sieciowej, czy konfiguracja i nadzór nad filtrami sieciowymi. Ciekawym zastosowaniem jest tutaj zdalna obsługa sprzętu sieciowego, routerów i firewalli, umożliwiając bezproblemową kontrolę całej sieci z jednego miejsca.

Działanie może być bardzo zbliżone do działania najpopularniejszego protokółu służącego do zarządzania sieciami – SNMP (*Simple Network Management Protocol*), który jest standardem używanym do nadzoru i zarządzania różnymi elementami sieci telekomunikacyjnych. Protokół SNMP zakłada istnienie w zarządzanej sieci dwóch rodzajów urządzeń: *zarządzających*, na których uruchomiony jest program menadżera SNMP i *zarządzanych*, na których działa program agenta SNMP. Cała istota działania oparta jest na odczytywaniu lub ustawianiu przez menadżera odpowiednich zmiennych u zarządzanych agentów [17][18].

Poprzez mechanizmy kontroli połączenia (`INIT`, `BYE` i `PING`) oraz dodatkowo pakiety statusowe, w łatwy sposób możemy nadzorować stan uruchomionych usług w sieci, lub innych narzędzi bądź systemów, a w razie wystąpienia jakichkolwiek problemów podjąć odpowiednie kroki, jak na przykład powiadomienie administratora lub zdalna zmiana konfiguracji.

Zastosowań zdalnego wykonywania procedur jest wiele, elastyczna budowa pakietu `COMMAND` sprzyja wykorzystaniu protokołu SLCP w najróżniejszych rozwiązaniach, nawet tych eksperymentalnych, wymagających kontroli i nadzoru, bądź zarządzania w sieciach lokalnych.

Niekoniecznie musi to być sprzęt komputerowy lub sieciowy, czy kontrola usług, oprogramowania sieciowego. Ciekawym zastosowaniem jest wykorzystanie protokołu i jego możliwości w komunikacji z innymi urządzeniami podłączonymi do sieci, pełniącymi różne funkcje użytkowe.

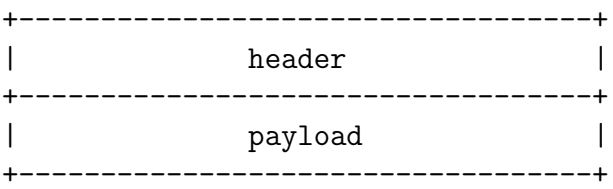
4. Specyfikacja

Secure LAN Communication Protocol dostarcza bezpiecznych metod komunikacji pomiędzy dwoma hostami (*host-to-host*), grupą hostów (*host-to-group*) oraz wszystkimi pozostałymi maszynami (*host-to-network*) w sieci lokalnej, bądź pojedynczym segmencie sieci.

Protokół ten oparty jest na datagramowej transmisji pakietów UDP, do komunikacji pomiędzy zdefiniowaną grupą hostów można wykorzystać technikę multicast i brodcast, a bezpieczeństwo danych oparto na algorytmie szyfrowania asymetrycznego RSA z kluczami 1024-bitowymi.

4.1. Format pakietu

Budowa pakietu protokółu SLCP, podobnie jak większości protokołów sieciowych, jest segmentowa. Składa się z dwóch bloków: nagłówka (*header*) i danych (*payload*).



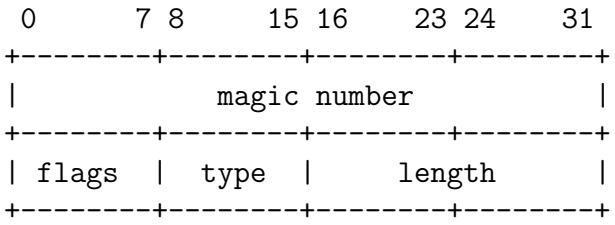
Rys. 4.1. Budowa pakietu

Specyfikacja ta definiuje kilka podstawowych typów pakietów protokółu SLCP:

`INIT`	informacja o włączeniu działania usługi na danym hoście
`BYE`	informacja o zakończeniu działania usługi na danym hoście
`PING`	informacja o aktywnym działaniu usługi
`STATUS`	informacja o statusie usługi
`MESSAGE`	precyzowana przez usługę wiadomość
`COMMAND`	wykonanie polecenia usługi lub innej czynności

4.1.1. Nagłówek

Format nagłówka pakietu protokołu SLCP jest niezmienny dla wszystkich typów pakietów. Zawiera on niezbędne informacje o pakiecie i danych jakie zostały w nim zawarte.



Rys. 4.2. Format nagłówka pakietu

Pola mają następujące znaczenie:

magic number: 32 bity

Stała wartość identyfikująca pakiet protokołu SLCP o wartości `0x50434C53`, która odpowiada ciągowi znaków `SLCP`.

flags: 8 bitów

Informacje określające właściwości danego pakietu. Aktualnie tylko 2 flagi są używane, (kolejno z lewej do prawej) oznaczają:

`ALL` pakiet wysłany do wszystkich hostów w danej sieci/segmencie

`CRP` dane zawarte w pakiecie są szyfrowane

type: 8 bitów

Jedna ze stałych określających typ pakietu:

`INIT`	`0`
`BYE`	`1`
`PING`	`2`
`STATUS`	`3`
`MESSAGE`	`4`
`COMMAND`	`5`

length: 16 bitów

Rozmiar całego pakietu w bajtach: nagłówek i dane. Minimalna długość to 8 bajtów (rozmiar samego nagłówka), a maksymalna długość to teoretyczny limit danych przenoszony przez datagram UDP, czyli 65,527 bajtów.

4.1.2. Dane

Format danych uzależniony jest od typu pakietu. Zależnie od ustawień flag w nagłówku dane zawarte w protokole mogą być zaszyfrowane w celu zapewnienia bezpieczeństwa

i poufności transmisji przed niepowołanymi próbami dostępu do nich.

Maksymalna długość danych jaka może być przenoszona przez pojedynczy pakiet protokołu SLCP to teoretycznie 65,519 bajtów.

Dokładny opis formatu danych poszczególnych typów pakietów oraz ich przeznaczenie i cel zostały opisane w następnym podrozdziale.

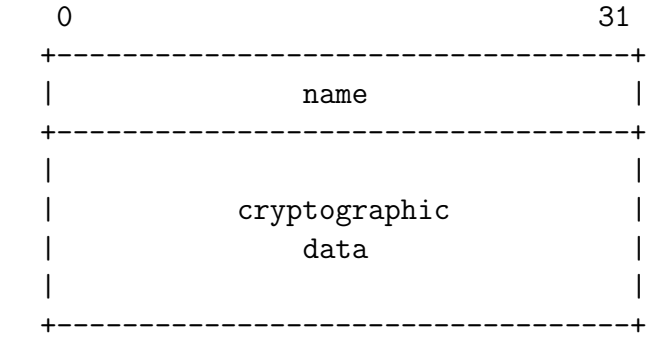
4.2. Typy pakietów

W zamieszczonych poniżej opisach poszczególnych typów pakietów protokołu, w celu łatwiejszego przedstawienia budowy i formatu danych, pominięto nagłówek, który dla wszystkich pakietów ma ten sam kształt, a przedstawiany tutaj mógłby wprowadzać małe zamieszanie.

Wszelkie rozmiary pól podano w bajtach.

4.2.1. INIT

Pakiet `INIT` informuje o włączeniu usługi na danym hoście, jest wysłany do wszystkich pozostałych hostów podłączonych do danej sieci. Zaraz za pakietem `INIT` zalecane jest wysłanie także pakietu `STATUS`.



Rys. 4.3. Format danych pakietu INIT

Pola mają następujące znaczenie:

name: 32 bajty

Ciąg znakowy określający nazwę identyfikującą hosta na którym uruchomiono usługę. Może to być dowolna nazwa. Maksymalna długość to 32 znaki, w przypadku mniejszych nazw ciąg powinien zostać zakończony znakiem NULL w konwencji ciągów znakowych języka C.

cryptographic data: 128 bajtów

Dane kryptograficzne w postaci klucza publicznego RSA niezbędne do zaszyfrowania

danych zawartych w pakietach wysyłanych do danego hosta.

Odbiorca pakietu `INIT` zobowiązany jest do odesłania własnego pakietu `INIT` bezpośredni do hosta adresata, ale tylko w odpowiedzi na odebranie pakietu skierowanego do wszystkich hostów w sieci. W przeciwnym wypadku, odpowiedź na każdy pakiet `INIT` stworzy niekończącą się pętlą wysyłanych i odbieranych pakietów `INIT`.

4.2.2. BYE

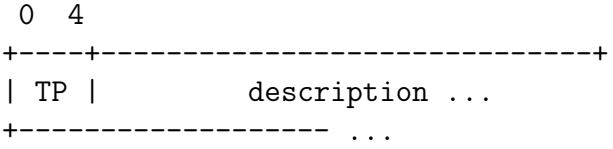
Pakiet `BYE` informuje o zakończeniu świadczenia usługi na danym hoście. Pakiet ten nie przekazuje żadnych danych, jedynie sam nagłówek z odpowiednim typem pakietu. Wysyłany jest do wszystkich hostów podłączonych do danej sieci

4.2.3. PING

Pakiet `PING` informuje pozostałe hosty o aktywnej uruchomionej usłudze na danym hoście. Pakiet ten nie przekazuje żadnych dodatkowych danych, jedynie sam nagłówek z ustawionym typem pakietu. Wysyłany jest do wszystkich hostów podłączonych do danej sieci co 60 sekund, począwszy od wysłania pakietu `INIT`. W przypadku nie otrzymania przez pozostałe hosty, na których uruchomiono usługę, pakietu `PING` w czasie nie później niż 75 sekund od ostatniego pakietu `PING` lub `INIT`, host zostaje uznany za nieosiągalny i usunięty z listy aktywnych hostów danej usługi działających w sieci.

4.2.4. STATUS

Pakiet `STATUS` służy do informowania innego hosta, bądź wszystkich pozostałych o statusie usługi na danym hoście, gdzie została uruchomiona.



Rys. 4.4. Format danych pakietu STATUS

Pola mają następujące znaczenie:

type (TP): 4 bajty

Liczba 32-bitowa określająca typ statusu. Protokół SLCP definiuje kilka standardowych typów statusu:

`AVAILABLE`	1	dostępny i gotowy do działania
`AWAY`	2	chwilowo nieosiągalny
`BUSY`	3	zajęty, wykonujący jakieś działanie/operacje

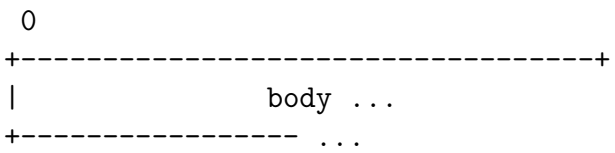
description:

Opcjonalny ciąg znakowy zawierający dodatkowe, opisowe dane statusu lub jakiekolwiek inne dane binarne związane z statusem.

Długość treści opisowej obliczana jest na podstawie danych zawartych w nagłówku (długość pakietu – długość nagłówka – 4 bajty). Minimalna długość wynosi 0 bajtów, a maksymalna liczbie danych jakie może zmieścić pakiet protokołu SLCP, pomniejszony o 4 bajty, czyli 65,515 bajtów.

4.2.5. MESSAGE

Pakiet `MESSAGE` służy do transmisji ściśle nie zdefiniowanych przez protokół wiadomości do pojedynczego lub wszystkich hostów podłączonych do sieci. Format przesyłanej treści zależy od rodzaju usługi bazującej na protokole SLCP.



Rys. 4.5. Format danych pakietu MESSAGE

Pola mają następujące znaczenie:

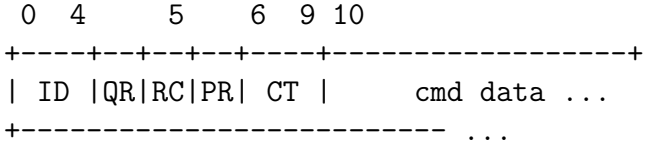
body:

Treść wiadomości.

Długość treści obliczana jest na podstawie danych zawartych w nagłówku. Minimalna długość wynosi 0 bajtów, a maksymalna liczbie danych jakie może zmieścić pakiet protokołu SLCP - 65,519 bajtów.

4.2.6. COMMAND

Pakiet `COMMAND` służy do wykonywania zdalnych procedur i poleceń. Wykorzystywany jest w roli zapytania inicjującego wykonanie procedury oraz odpowiedzi zwrotnej zawierającej wynik wykonania procedury.



Rys. 4.6. Format danych pakietu COMMAND

Pola mają następujące znaczenie:

identifier (ID): 4 bajty

Liczba 32-bitowa identyfikująca pakiet, niezbędna do jednoznacznego powiązania zapytania i odpowiedzi. Tworzona przez aplikację wysyłającą zapytanie, odbiorca przepisuje identyfikator do odpowiedzi.

query or response (QR): 1 bit

Bit określający, czy pakiet jest zapytaniem czy odpowiedzią:

query	`0`
response	`1`

return code (RC): 3 bity

Zwracany kod z wykonania procedury, przyjmujący jedną z kilku zdefiniowanych, stałych wartości:

`EOK`	`0`	brak błędu, procedura wykonana pomyślnie
`EFAULT`	`1`	błąd wykonania procedury
`ENODATA`	`2`	brak wymaganych danych (argumentów)
`EBUSY`	`3`	host, aplikacja zajęta
`ECANCELED`	`4`	operacja anulowana
`ENOEXIST`	`5`	procedura nie istnieje
`ENOTSUPP`	`6`	procedura nie zaimplementowana

priority (PR): 4 bity

Liczba 4-bitowa określająca priorytet wykonania procedury. Im większa wartość tym wyższy priorytet.

cmd type (CT): 4 bajty

Liczba 32-bitowa określająca typ polecenia lub procedury do wykonania (zapytanie) lub wykonanej (odpowiedź).

cmd data:

Opcjonalne dane, w postaci ciągu danych binarnych, przekazywane do procedury (zapytanie) lub wynik operacji (odpowiedź), odpowiednio interpretowane przez aplikację usługi.

Długość opcjonalnych danych obliczana jest na podstawie danych zawartych w nagłówku. Minimalna długość wynosi 0 bajtów, a maksymalna 65,510 bajtów.

5. Biblioteka slcp

Nic tak nie rewiduje przychodzących pomysłów niż ich równoległa implementacja i możliwość przetestowania. Sam Bjarne Stroustrup pracując nad językiem C++, a dokładniej to nad jego pierwowzorem – C z klasami, najpierw implementował w swoim kompilatorze CFront nowe pomysły i rozszerzenia języka. A dopiero później po przetestowaniu i zrewidowaniu pomysłu był gotowy na wprowadzenie danej funkcjonalności i rozwiązania do projektowanego języka [19].

Biblioteka slcp jest w pełni funkcjonalną biblioteką protokołu SLCP napisaną w języku C++. Pozwala na łatwe tworzenie aplikacji i usług bazujących na protokole SLCP, a także na integrację komunikacji i możliwości oferowanych przez SLCP w istniejących aplikacjach. Dzięki elastycznej, obiektowej budowie, bardzo łatwo daje się rozszerzać o nowe możliwości i rozszerzenia protokołu.

Biblioteka licencjonowana jest na zasadach GNU GPL w wersji drugiej.

5.1. Funkcje i możliwości

Biblioteka slcp implementuje prawie w pełni możliwości i funkcje jakie oferuje protokół SLCP z wyjątkiem szyfrowania przesyłanych danych.

Obsługiwane elementy:

- inicjacja i podtrzymanie połączenia
- powiadomienia o aktywacji usługi, hostów w sieci
- odbieranie oraz wysyłanie statusów i wiadomości
- obsługa zdalnych procedur

Prócz obsługi samego protokołu SLCP, biblioteka posiada kilka ciekawych możliwości ułatwiających jej wykorzystanie. Do najważniejszych można zaliczyć elastyczny system obsługi zdarzeń.

5.2. Budowa i architektura

Architektura biblioteki slcp oparta jest o wzorzec projektowy obserwator[1]. Wykorzystywany on jest w slcp jako system zdarzeń, przez co zachowujemy odseparowanie działania samej biblioteki od konkretnej implementacji zachowania, czynności jakie powinny zostać wykonane przy zajściu danego zdarzenia protokołu, bądź innych okolicznościach.

Wszelkie typy i funkcje definiowane przez bibliotekę znajdują się w przestrzeni nazw `slcp`.

5.2.1. Obsługa zdarzeń

Działanie jest bardzo proste, kiedy zachodzi jakieś zdarzenie powiadamiane zostają wszyscy obserwatorzy „podłączeni" do danego zdarzenia. System iteruje tablice obserwatorów (handlerów) jakie zostały zarejestrowane dla zdarzenia i wykonuje na rzecz tych obiektów odpowiednie metody zależne od zaistniałych okoliczności. W dowolnym czasie można dla danego zdarzenia podłączyć lub odłączyć obserwatora. Ilość podłączonych obserwatorów do jednego zdarzenia jest nieograniczona, dzięki temu można zaimplementować kilka odizolowanych od siebie interpretacji i poleceń działających na zaistnienie danej, określonej okoliczności (zdarzenia).

Obserwatorem używanym przez bibliotekę slcp mogą zostać dowolne klasy implementujące odpowiedni dla danego typu zdarzeń interfejs, czyli dziedziczące po czysto wirtualnych klasach abstrakcyjnych i implementujące odpowiednie metody wirtualne.

Obecnie istnieje 5 typów handlerów:

ConnectionHandler

Interfejs do implementacji obserwatora powiadamianego o stanie i zdarzeniach związanych z połączeniem aplikacji klienta. Po nawiązaniu polaczenia (wysłaniu pakietu `INIT`) wywoływana jest metoda `HandleConnect()`, przy rozłączeniu (po wysłaniu `BYE`, lub wystąpieniu innego problemu z połączeniem) - `HandleDisconnect()` oraz dodatkowo `HandlePing()` po wysłaniu informacji „alive" (pakietu typu `PING`).

ConnectionDataHandler

Klasa bazowa do implementacji obserwatora połączenia, śledząca ruch sieciowy - odbierane i wysyłane dane. Metody `HandleIncomingData()` i `HandleOutgoingData()` wywoływane są odpowiednio, przy odebraniu pakietu (przed powiadomieniem odpowiednich obserwatorów) i przy wysyłaniu pakietu (przed fizycznym wysłaniem).

[1] Znany rownież jako wzorzec *observer* lub *listener*.

HostConnectionHandler

Interfejs służący do implementacji obserwatora otrzymującego notyfikacje o nawiązaniu lub zerwaniu połączenia danego hosta w sieci. Posiada te same metody co interfejs `ConnectionHandler`, ale uruchamiane przy odebraniu pakietów `INIT`, `BYE` i `PING` w kontekście konkretnego hosta.

StatusHandler

Wirtualny interfejs do implementacji obserwatora powiadamianego o zmianie statusu. Po odebraniu pakietu `STATUS` wywoływana jest metoda `HandleStatus()`.

MessageHandler

Interfejs służący do otrzymywania wiadomości, pakietów typu `MESSSAGE`. Po odebraniu wiadomości zostaje wywoływana metoda `HandleMessage()`.

CommandHandler

Interfejs obserwatora dla wywoływanych procedur zdalnych. Uruchamia metodę `HandleCommand()` po otrzymaniu pakietu `COMMAND`.

Wszystkie klasy dziedziczące po tych interfejsach mogą być rejestrowane jako handlery odpowiednich typów w obiekcie będącego instancją klasy `Client`.

5.2.2. Klasa Client

Klasa `Client` implementuje podstawową funkcjonalność klienta protokołu SLCP, a tym samym będąca klasą bazową dla aplikacji. Aby z niej skorzystać należy utworzyć obiekt tego typu z podaniem odpowiednich danych do nawiązania połączenia, w konstruktorze, bądź korzystając z metod typu „setter". Następnie można zarejestrować odpowiednie obiekty obsługi zdarzeń i wywołać metodę `Connect()`, która inicjuje nawiązanie połączenie oraz tworzy pętlę nasłuchującą sieć i wyzwalającą odpowiednie mechanizmy notyfikacji podłączonych obserwatorów.

Wywołanie funkcji `Connect()` jest blokujące, nawiązanie połączenia i nasłuchiwanie sieci blokuje dalsze wykonywanie funkcji znajdujących się po uruchomianiu `Connect()`. Funkcja ta kończy swoje działanie dopiero przy rozłączeniu polaczenia. Nie jest to wielkim problemem, ponieważ dalsza obsługa jest skupiona wokół obserwatorów i funkcji obsługi zdarzeń implementujących odpowiednie zachowanie aplikacji.

Mechanizmy nieblokujące mogą być bardzo przydatne w niektórych zastosowaniach, więc w przyszłych wersjach biblioteka slcp powinna się o nie wzbogacić.

5.2.3. Reprezentacja pakietów

W interfejsie publicznym udostępnianym przez bibliotekę nie mamy żadnych metod bezpośredniego dostępu do pakietów protokołu SLCP.

Biblioteka definiuje abstrakcyjne reprezentacje pakietów `MESSAGE`, `STATUS` i `COMMAND` w postaci odpowiednich klas dla każdego typu: `Message`, `Status` oraz `Command`. Klasy te implementują interfejs `Packet` oraz zawierają specyficzne dla danego pakietu metody dostępowe. Za pomocą metody `Client::Send()` mogą być wysyłane w sieć.

Pakiety protokołu SLCP typu `INIT`, `BYE` i `PING` są obsługiwane wewnętrznie przez samą bibliotekę. O ich otrzymaniu, bądź wysłaniu możemy dowiedzieć się jedynie z odpowiednich handlerów związanych z połączeniem naszego klienta (`ConnectionHandler`) oraz innych hostów w sieci (`HostConnectionHandler`).

Klasa `RawPacket` reprezentuje cały surowy pakiet protokołu. Umożliwia ona łatwy dostęp do poszczególnych bloków pakietu - nagłowka (`PacketHeader`) i danych (`PacketData`) bez potrzeby jawnej konwersji typów. Pracując w roli *wrappera* dla surowych danych, eliminujemy zbędne kopiowanie danych, mając łatwy dostęp do segmentów pakietu, co bardzo upraszcza proces przetwarzania danych otrzymanych z sieci.

Klasa ta jest również użyteczna przy tworzeniu surowego pakietu do wysyłki. Wewnętrznie alokuje wymagane miejsce na surowy pakiet, a dzięki wspomnianym metodom dostępu do poszczególnych segmentów pakietu, łatwo daje się wypełnić danymi. Wewnętrzna alokacja daje dodatkowe korzyści i bezpieczeństwo, zwalnia nas z obowiązku ręcznej dealokacji zasobów, a także wykorzystując idiom RAII (*Resource Acquisition Is Initialization*) jest bezpieczna w różnych sytuacjach wyjątkowych.

Surowe dane pakietu zwracane z obiektu typu `RawPacket` są konwertowane przez odpowiednie klasy abstrakcyjnej reprezentacji poszczególnych pakietów. Konwersja ta zachodzi w odpowiednim konstruktorze konwersji typu, przyjmującym surowe dane pakietu.

Jak już wspomniano, klasy reprezentujące pakiety implementują interfejs `Packet`. Interfejs ten definiuje kilka standardowych metod jakie są potrzebne do konwersji w obie strony, zatem klasy te posiadają rownież mechanizmy do konwersji w druga stronę, czyli z klasy reprezentującej pakiet do surowej postaci danych pakietu.

5.3. Kompilacja

System budowania biblioteki opary jest na systemie bakefile[2]. Narzędzie to służy do generowania dedykowanych dla konkretnego systemu i środowiska odpowiednich plików reguł makefile i plików projektów do popularnych IDE, bazując na opisanym w formacie XML

[2] `http://www.bakefile.org`

projekcie. Jest to idealne narzędzie dla oprogramowania multi-platformowego, przy zmianach zachodzących w projekcie, wymagających uaktualnień w plikach przeznaczonych do procesu budowania, nie musimy ręcznie wprowadzać tych poprawek we wszystkich plikach makefile dla obsługiwanych systemów i środowisk, gdzie łatwo można byłoby popełnić błąd.

Pliki dla bakefile dostępne są w katalogu `build/bakefiles` w źródłach biblioteki. Na ich podstawie możemy wygenerować odpowiednie, docelowe pliki pozwalające na zbudowanie biblioteki. Wygenerować je możemy uruchamiając bakefile z poziomu tego katalogu, wraz z odpowiednimi opcjami:

```
Bakefile -f format file.bkl
```

Parametr `format` określa format wyjściowego pliku, czyli jakiego rodzaju chcemy otrzymać plik i dla jakiego systemu, środowiska, czy ma to być plik dla makefile, czy może plik projektu dla jakiegoś popularnego IDE. Dostępne formaty można znaleźć w pomocy (przełącznik `--help`) lub w dokumentacji.

Od czasu kiedy zaczniemy używać bakefile w swoich projektach do generowania makefile w wielu różnych formatach możemy znużyć się wklepywaniem kilku podobnych poleceń, aby wygenerować wszystkie potrzebne nam pliki. Można napisać jakiś skrypt automatyzujący te czynności, ale jest inny, lepszy sposób.

W pakiecie z bakefile znajduje się przydatne narzędzie służące do wsadowego generowania bakefile - `bakefiles_gen`. Narzędzie to bazuje na pliku `Bakefiles.bkgen`, który w prostym XML-owym formacie opisuje w jakim formacie co i jak wygenerować z plików bakefile. Użycie jest proste, wystarczy wejść do katalogu gdzie znajduje się plik `Bakefiles.bkgen` i wykonać polecenie:

```
bakefile_gen
```

Odpowiednie pliki zostaną wygenerowane automatycznie.

Plik dla wsadowego generowania bakefile dla biblioteki slpc znajduje się w katalogu wraz z plikiem bakefile, czyli `build/bakefiles`.

W katalogu `build` można znaleźć wygenerowane pliki makefile dla kompilatora GNU GCC pod systemy uniksowe oraz windowsowego portu - MinGW, a także pliki makefile i pliki projektu dla Microsoft Visual C++ 2008. Przydatne dla tych co nie posiadają lub nie używają narzędzi bakefile.

Sam proces kompilacji i budowania biblioteki nie rożni się niczym szczególnym od typowych metod.

Aby skompilować bibliotekę pod systemem unikso-podobnym, znajdując się w katalogu ze źródłami, wystarczy wydać polecenie:

```
make
```

Pod systemem Windows znajdując się w katalogu `build`, proces kompilacji przy użyciu

plików makefile jest bardzo podobny.

Dla MInGW, wykorzystujemy narzędzie make dostarczane z tym pakietem:

```
mingw32-make -f makefile.gcc
```

Analogicznie jest w przypadku środowiska Visual Studio, korzystamy z narzędzia make i pliku makefile odpowiedniego dla tego środowiska:

```
nmake -f makefile.vc
```

Jak wspomniano, istnieją pliki projektu dla tego środowiska, więc można bezpośrednio spod IDE skompilować bibliotekę.

Po wykonaniu podanych poleceń, w odpowiednim podkatalogu (zależnie od rodzaju kompilacji) w katalogu `lib` powinien znaleźć się plik wynikowy biblioteki - `slcp` z rozszerzeniem odpowiednim dla trybu kompilacji i systemu.

Nie ma żadnych skryptów ani sekcji „install" w plikach makefile, zatem chcąc „zainstalować" bibliotekę w systemie, aby móc z niej korzystać w tworzonym oprogramowaniu, musimy dokonać tego ręcznie. Najlepiej pliki nagłówkowe skopiować do katalogu o nazwie `slcp` umiejscowionego wśród pozostałych nagłówków innych bibliotek, zależnie od systemu. Analogicznie z plikiem wynikowym.

W przypadku systemu Windows i VC++, wystarczy w opcjach dodać ścieżki do odpowiednich katalogów.

5.4. Dokumentacja

Dokumentacja biblioteki, w języku angielskim, zawierającą w głównej mierze referencje i opis funkcji oraz klas, zawarta jest w źródłach oprogramowania. Pliki nagłówkowe biblioteki zawierają komentarze w formacie zgodnym z Doxygen[3].

Z pomocą narzędzia Doxygen możliwe jest wygenerowanie dokumentacji w postaci bardziej strawnych dla ludzi formatach jak HTML, LaTeX, RTF, PDF, czy wielu innych obsługiwanych przez to narzędzie. Dodatkowo w przypadku projektu zorientowanego obiektowo automatycznie można wygenerować także graficzne diagramy i grafy wizualizujące relacje i zależności pomiędzy poszczególnymi klasami i plikami projektu.

Doxygen analizuje zawartość plików źródłowych szukając odpowiednio oznaczonych i sformatowanych komentarzy, aby na ich podstawie utworzyć wynikową dokumentację. Dzięki wykorzystaniu informacji zawartych bezpośrednio w kodzie źródłowym, zachowujemy i utrzymujemy na wysokim poziomie spójność i zgodność dokumentacji z faktycznym stanem oprogramowania.

Dokumentację można wygenerować wydając z linii poleceń komendę:

```
doxygen [configName]
```

[3] `http://www.doxygen.nl`

Jako parametr podajemy ścieżkę do pliku zawierającego odpowiednią konfigurację dla daneo projektu, standardowo plik ten nosi nazwę *Doxyfile*. W przypadku biblioteki slcp, plik ten z predefiniowanymi ustawieniami można znaleźć w katalogu `doc`.

Modyfikując plik konfiguracyjny możemy dostosować tworzoną dokumentacje wedle własnych preferencji i upodobań, w preferowanym formacie.

Wygenerowana dokumentacja w formacie HTML rozpowszechniana jest wraz z źródłami biblioteki w katalogu `doc/html`.

5.5. Modyfikacje i rozszerzenia

Rozszerzalność i edycja biblioteki jest bardzo prosta.

W głównej mierze zawdzięczamy to dobremu systemowi obsługi zdarzeń, bazującemu na wzorcu projektowym obserwatora oraz projektowi modelu obiektowego biblioteki.

Bardzo łatwo można rozszerzyć bibliotekę o obsługę nowych rozszerzeń i funkcji samego protokołu SLCP, w głównej mierze o nowe pakiety.

Dzięki interfejsowi `Packet`, wystarczy stworzyć nową klasę reprezentującą dodawany pakiet oraz implementację metod zawartych w tym interfejsie. A bez problemy biblioteka zacznie „z marsza" obsługiwać wysyłanie nowych pakietów. W przypadku odbierania i przetwarzania pakietów musimy zaingerować w funkcję `Client::handleRecvData()` dodając odpowiedni warunek do wyrażenia switch. Prócz tego należy stworzyć nowy interfejs handlera dla implementacji obserwatora otrzymującego notyfikacje związane z nowym pakietem, oczywiście o ile jest to nam potrzebne i odpowiednio go powiadamiać o zaistniałych zdarzeniach.

Po analizie struktury i kodu biblioteki przekonamy się, że rozszerzalność biblioteki, jej modyfikacja, dostosowywanie do własnych potrzeb oraz nowych możliwości jest bardzo prosta i nie wymaga dużego nakładu pracy.

5.6. Rozwój

Biblioteka jest ciągle rozwijana, autor zamierza w miarę możliwość rozszerzać ją o nowe funkcje i mechanizmy mogące się przydać w jej wykorzystaniu, a także o nowe możliwości protokołu SLCP.

W głównej mierze wśród pozycji listy rzeczy planowanych do zrobienia, podstawową kwestią jest zaimplementowanie pełnej obsługi protokołu SLCP. Aktualnie brakuje wsparcia dla szyfrowania, więc ta kwestia powinna posiadać najwyższy priorytet.

Z pozostałych planowanych usprawnień i dodatków:

- mechanizmy nieblokującej obsługi połączeń i nasłuchiwania sieci
- możliwość zablokowania przetwarzania odebranego i wysyłanego pakietu

Implementacji wspomnianych wyżej pozycji usprawnień i dodatków można spodziewać się w przyszłych wersjach biblioteki.

5.7. Przykłady użycia

Poniższy podrozdział demonstruje proste przykłady użycia biblioteki slcp. Zamieszczone tutaj przykłady w głównej mierze oparto na implementacji prostych usług sieciowych, aby można było skupić się na ukazaniu podstawowych technik pracy z biblioteką, sposobów użycia oraz jej walorów.

Prezentowane przykłady pokazują jak łatwa i prosta, z pomocą biblioteki slcp, staje się obsługa protokołu SLCP. Nie musimy wnikać w szczegóły implementacyjne i sieciowe, aby móc w pełni korzystać z oferowanych przez bibliotekę i protokół dobrodziejstw.

Pełne kody źródłowe przedstawionych tutaj programów dostępne są wraz z źródłami biblioteki, w katalogu `examples`.

5.7.1. Usługa echo

Poniższy przykład jest bardzo prosty, demonstruję użycie biblioteki i protokołu SLCP do zaimplementowania usługi ECHO.

Usługa ECHO jest internetowym protokołem zdefiniowanym w dokumencie RFC 862 [20]. Jej pierwotnym celem było testowanie i pomiary trasy pakietów w czasach kiedy sieci komputerowe raczkowały [21]. Obecnie do tego celu używa się protokołu ICMP, korzystając z aplikacji *traceroute* i *ping*.

Usługa ta pracuje domyślnie na porcie 7 protokołów TCP i UDP. Jej działanie jest bardzo proste, odsyła do nadawcy w niezmienionej postaci wszelkie odebrane dane.

Działanie naszej wersji usługi ECHO opartej na bibliotece slcp jest bardzo podobne. Wysłanie wiadomości do hosta, gdzie uruchomiono usługę będzie skutkować natychmiastowym jej odesłaniem do adresata, w ten sposób „usłyszymy” echo naszego komunikatu.

Nasza aplikacja musi reagować na otrzymywane wiadomości, zatem niezbędna jest implementacja interfejsu `MessageHandler`. Nasza główna klasa aplikacji może dziedziczyć bezpośrednio z tego interfejsu:

```
class Echo : public MessageHandler {...};
```

Odsyłamy przychodzące wiadomości, więc aby to spełnić musimy mieć łatwy dostęp do instancji obiektu `Client`. Przez co najlepszym i prostym rozwiązaniem jest zawarcie obiektu klienta w klasie handlera. Oczywiście istnieje wiele innych metod umożliwiających osiągnięcie celu. Mogliśmy główny obiekt dziedziczyć nie tylko po interfejsie `MessageHandler`, ale także po klasie `Client`. My akurat wybraliśmy tą metodę.

W konstruktorze klasy `Echo` inicjalizujemy tworzony obiekt klienta wymaganymi informacjami oraz rejestrujemy naszą klasę obsługi zdarzeń wiadomości:

```
Echo(unsigned short port)
    : m_client(port)
{
    Status status(Status::Available, "echo service...");
    m_client.SetName("echo");
    m_client.SetStatus(status);
    m_client.RegisterMessageHandler(this);
}
```

Mając dostęp do instancji klienta, reimplementacja wirtualnej funkcji `HandleMessage()` odbierającej pakiety wiadomości jest banalna:

```
void HandleMessage(HostId host, const Message& message)
{
    m_client.Send(message, host);
}
```

Dodajemy jeszcze niezbędną metodę umożliwiającą rozpoczęcie pracy programu, czyli nawiązanie przez klienta połączenia z siecią:

```
void Start()
{
    m_client.Connect();
}
```

A cały kod zawarty w funkcji `main` ogranicza się tylko do 2 linijek - stworzenie obiektu klasy `Echo` i wywołanie metody uruchamiającej działanie usługi:

```
int main(int argc, char* argv[])
{
    Echo echo(10007);
    echo.Start();

    return 0;
}
```

Jak widać, dzięki wspaniałej obsłudze zdarzeń, tworzenie aplikacji korzystających z protokołu SLCP z zastosowaniem niniejszej biblioteki jest bardzo proste i przyjemne.

6. Program Lanek

Program *Lanek* jest prostą aplikacją realizującą komunikację tekstową pomiędzy użytkownikami sieci lokalnej. Jest to prosta realizacja jednego ze scenariuszy przedstawionych w rozdziale 3 do jakich można wykorzystać protokół SLCP.

Aplikacja została napisana w języku C++ z wykorzystaniem biblioteki slcp oraz toolkitu graficznego wxWidgets. Dzięki zastosowaniu multiplatformowych rozwiązań, aplikacja jest przenośna miedzy platformami, oznacza to tyle, że bez żadnych większych problemów można skompilować i używać program Lanek pod dowolną platformą obsługiwaną przez obie biblioteki.

Mimo, iż sam program jest dosyć użyteczną aplikacją, posiada ograniczone możliwości i funkcje, bo jego głównym celem była możliwość przetestowania biblioteki slcp oraz prezentacja możliwości protokołu SLCP. Z tych powodów autor nie przewiduje dalszego rozwoju aplikacji, ale mimo to nic nie stoi na przeszkodzie, aby program został rozwijany i wzbogacany o nowe funkcje przez zainteresowanych użytkowników. Więcej szczegółów na ten temat można znaleźć w końcowej części niniejszego rozdziału.

6.1. Budowa i architektura

Biblioteka slcp i wxWidgets zostały napisane w stylu obiektowym, pomijamy tutaj dywagacje na temat paradygmatu obiektowego i stopnia w jakim obie biblioteki w rzeczywistości zgodne są z założeniami obiektowego programowania. Aczkolwiek styl w jakim zostały napisane rzutuje na budowę i architekturę samej aplikacji.

Budowa programu jest bardzo prosta, można uznać, że to typowe, modelowe wykorzystanie użytych bibliotek.

Aplikacja działa wielowątkowo, na dwóch watkach. Jeden z nich, główny watek jest odpowiedzialny za interfejs użytkownika, w którym wykonywane są wszelkie polecenia i prace związane z obsługą interfejsu oraz systemem zdarzeń (eventów), na którym opiera się działanie i architektura aplikacji korzystających z wxWidgets. Drugi wątek związany jest z biblioteką slcp, kontroluje on cześć sieciową aplikacji, nasłuchując przychodzących danych.

Komunikacja pomiędzy tymi wątkami oparta jest na systemie eventów. Głównym spoiwem jest klasa `SlcpClient`, która nie tylko jest klasą dziedziczną `slcp::Client`, a tym samym klientem protokołu SLCP, ale także implementuje wszystkie handlery biblioteki, konwertując sygnały i zdarzenia z biblioteki sieciowej na zdarzenia wxowe - oparte na `wxEvent`. A następnie przekazuje je do głównego wątku aplikacji.

Niezbędne jest takie przekazywanie sygnałów, nie możemy bezpośrednio z handlerów operować na funkcjach i metodach związanych z obiektami GUI. Operacje takie mogą być wykonane tylko z głównego wątku programu, gdzie stworzono instancję klasy `wxApp`.

Wszystkie elementy interfejsu użytkownika, takie jak okna, menu oparte są na XML-owym systemie zasobów – XRC (*XML Based Resource System*) dostępnym w wxWidgets. Wygląd poszczególnych elementów zapisany jest w plikach tekstowych w formacie XML. W czasie procesu budowania, pliki te wraz z przypisanymi grafikami kompresowane są do archiwum zip, tworząć plik zasobów XRS[1]. Pliki ten jest ładowany przy stracie programu i na jego podstawie tworzone są odpowiednie okna i inne elementy GUI.

6.2. Kompilacja i instalacja

System budowania aplikacji, podobnie jak biblioteki slcp i innego oprogramowania opartego na toolkicie wxWidgets, oparty jest na systemie bakefile.

Pliki dla bakefile dostępne są wraz ze źródłami aplikacji w katalogu `build/bakefiles`. Na ich podstawie możemy wygenerować odpowiednie, docelowe pliki pozwalające na zbudowanie programu.

W katalogu `build` znajdują się wygenerowane pliki dla systemu budowania opartego na plikach reguł `Makefile`. Są to pliki dla kompilatora GNU GCC pod systemy uniksowe, jego portu pod Widnows (MinGW) oraz dla kompilatora Visual C++ firmy Microsoft. Dodatkowo zawarte są również pliki projektu dla tego ostatniego środowiska.

Proces budowania jest podobny do tego opisanego w rozdziale dotyczącym biblioteki slcp i nie różni się niczym od typowego procesu budowania aplikacji przewidzianego dla konkretnego systemu i środowiska. Do prawidłowego przejścia tego procesu niezbędna jest obecność w systemie biblioteki slcp oraz wxWidgets.

Aplikacja nie wymaga instalacji. Można uruchamiać ją bezpośrednio dowolnego miejsca w systemie z plików wynikowych procesu budowania.

Rozpowszechnianie aplikacji ogranicza się do skopiowania dwóch plików: pliku wykonywalnego oraz pliku zasobów - `resources.xrs`, niezbędnego do prawidłowego działania programu. Oba pliki powinny znajdować się w tym samym katalogu.

[1]Wykorzystanie jednego pliku XRS jest lepszą opcją w porównaniu do luźnych plików XRC i grafik. Wygodę tą docenia się szczególnie przy rozpowszechnianiu aplikacji.

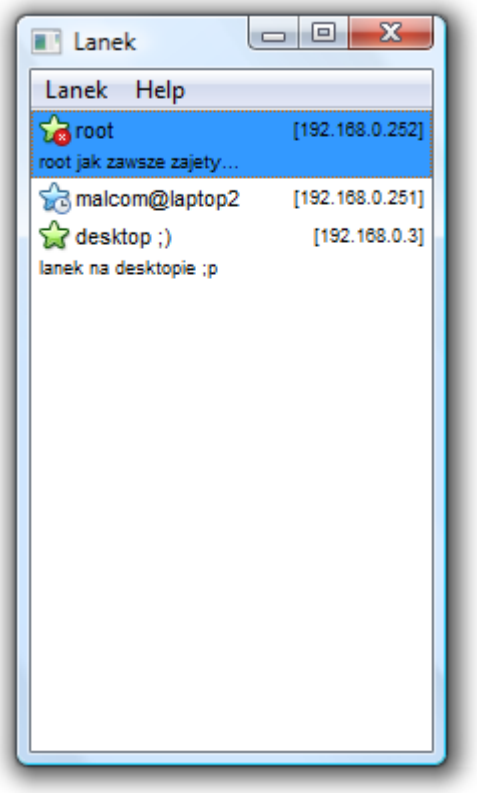
Jeśli, w czasie budowania linker dokonał statycznego linkowania bibliotek slcp i wxWidgets[2], żadne dodatkowe pliki nie są potrzebne do uruchomiania i działania aplikacji. Jedynie przy użyciu kompilatora VC++, zależnie od ustawień, może być wymagana instalacja pakietu *Redistributable Package*, zawierającego biblioteki uruchomieniowe (CRT).

6.3. Opis programu

Zamieszczone w tym podrozdziale rysunki w postaci zrzutów ekranów przedstawiających poszczególne elementy programu pochodzą z uruchomionej aplikacji pod kontrolą systemu Windows Vista oraz Ubuntu.

6.3.1. Okno główne

Po uruchomieniu programu Lanek naszym oczom ukaże się główne okno aplikacji. Okno to podobne jest do każdej innej okienkowej aplikacji, zawiera menu, dzięki któremu możemy wpływać na działanie samego programu, a także listy kontaktów wyświetlanej w głównej części okna. Wygląd okna głownego aplikacji prezentuje rysunek 6.1.



Rys. 6.1. Główne okno programu Lanek

[2]Co jest bardzo zalecane w celu łatwego używania i rozpowszechniania aplikacji.

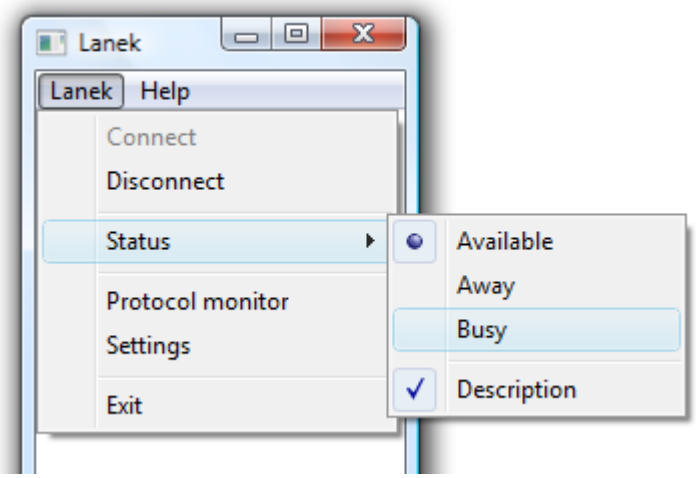
Lista kontaktów zajmująca centralną cześć okna zawiera pozycje, które reprezentują poszczególne kontakty, czyli pozostałych użytkowników w sieci używających programu Lanek. Obok nazwy kontaktu znajduje się ikona, która przedstawia aktualny stan użytkownika, a pod kontaktem widnieje opis statusu, jeśli dany użytkownik takowy sobie ustawił.

6.3.2. Status

Lanek pozwala na ustawienie jednego z kilku stanów, określających nasz aktualny stan, czy jesteśmy dostępni, czy zajęci. Statusy te są widoczne u każdego użytkownika w oknie głównym, gdzie obok kontaktu znajduje się ikona identyfikująca aktualny stan.

Dodatkowo możliwe jest do dodanie dowolnego opisu do ustawianego statusu, który pojawi się pod ikonką z nazwą kontaktu u pozostałych użytkownikach programu podłączonych do sieci. Opis ten może informować o naszej chwilowej nieobecności lub przekazywać dowolną wiadomość, np. że aktualnie jesteśmy nieosiągalni i będzie dostępni za 10 minut.

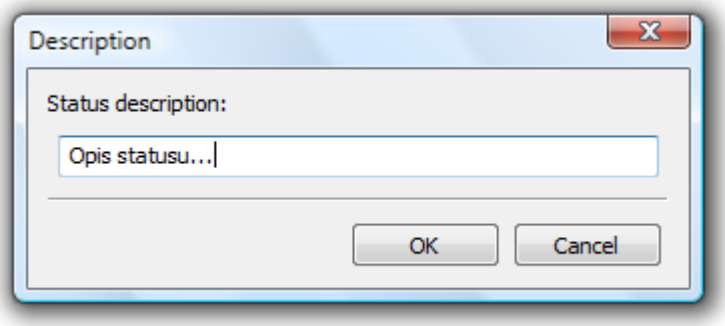
Zmianę aktualnego statusu można dokonać za pomocą menu, wybierając odpowiednią pozycję z podmenu `Lanek->Status`, co przedstawia rysunek 6.2.



Rys. 6.2. Menu wyboru statusu

Jeśli opcja *Description* w podmenu `Lanek->Status` jest zaznaczona, to po wybraniu odpowiedniego statusu z menu ukaże się dodatkowe okno dialogowe, jak na rysunku 6.3, proszące o wpisanie odpowiedniego opisu dla aktualnie ustawianego stanu.

Zmiana statusu zostanie wysłana do wszystkich uruchomionych aplikacji Lanek w sieci, przez co zmiana będzie widoczna przez pozostałych użytkowników.

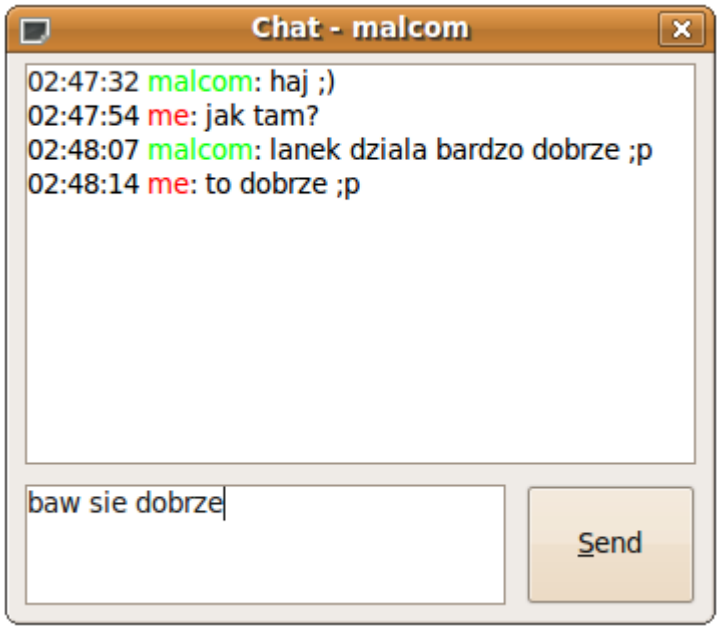


Rys. 6.3. Dialog ustawiania opisu statusu

6.3.3. Rozmowa

Głównym celem aplikacji jest możliwość prowadzenia rozmów z pozostałymi użytkownikami podłączonymi do sieci lokalnej, używającymi tej samej aplikacji.

Aby rozpocząć rozmowę z użytkownikiem klikamy dwukrotnie na jego pozycje na liście kontaktów, znajdującej się w głównym oknie programu. Skutkuje to wyświetleniem okna rozmowy, podzielonego na dwie części: cześć przedstawiającą historię konwersacji znajdującą się w górnej części okna oraz cześć odpowiadająca za redagowanie i wysyłanie wiadomości w dolnej. Widok okna rozmowy przedstawia rysunek 6.4.



Rys. 6.4. Okno rozmowy z użytkownikiem malcom

Wysłanie wpisanej wiadomości następuje po wciśnięciu klawisza `Enter` lub kliknięciu przycisku `Send`. Przejście do nowej linii w polu tekstowym wiadomości może być utrudnione ponieważ reakcją na naciśnięcie klawisza `Enter` jest wysyłanie, więc aby tego dokonać należy użyć kombinacji `Ctrl+Enter`.

Historia okna rozmowy jest pamiętana przez cały czas pracy aplikacji, zamknięcie i ponowne otwarcie okna nie ma na nią żadnego wpływu.

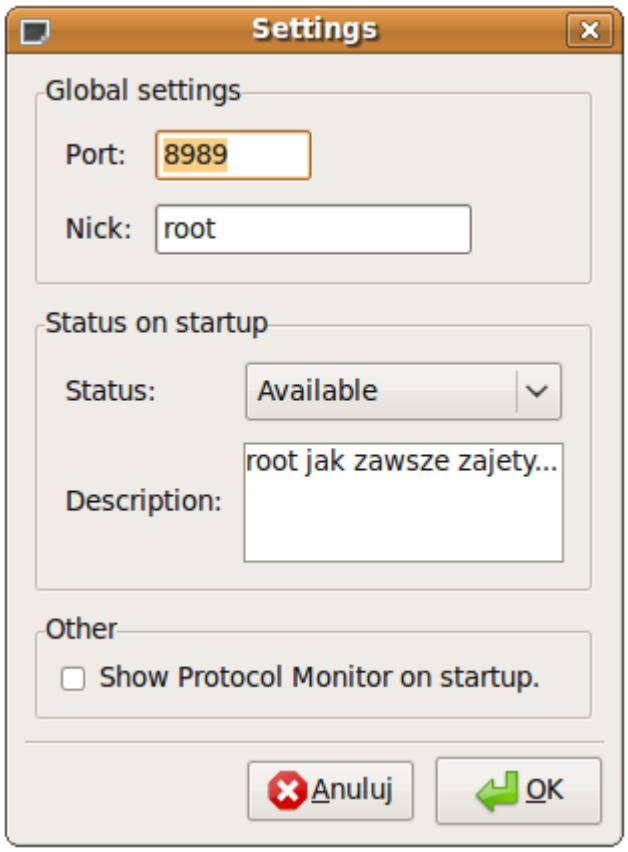
Jeśli w czasie nadejścia nowej wiadomości okno jest zamknięte lub ukryte to automatycznie zostanie otarte i wyświetlone na pierwszym planie, jeśli w tym czasie jest otwarte, ale znajduje się w tle, jest przykryte innymi oknami to, aby zwrócić uwagę użytkownika, okno na pasku zadań wykona efekt „flash”[3].

6.3.4. Opcje

Program Lanek umożliwia ustawnie kilku podstawowych opcji, takich jak używany przez aplikację numer portu, czy nazwa użytkownika pod jaką będzie widoczny użytkownik w sieci oraz opcje dotyczące statusu, automatycznie ustawianego przy starcie.

Biorąc pod uwagę prostotę i cel wykonania aplikacji, liczba dostępnych opcji konfiguracyjnych jest znikoma i ogranicza się tylko do podstawowych ustawień.

Aby zmienić opcje należy otworzyć okno ustawień, dokonać tego można przez wybranie odpowiedniej opcji z menu głównego okna: `Lanek->Settings`. Pojawi się okno dialogowe *Settings* podobne do tego z rysunku 6.5.



Rys. 6.5. Okno ustawień

[3]Efekt ten dostępny jest jedynie w portach wxWidgets dla Windows i GTK.

Po zmianie odpowiednich ustawień, należy kliknąć przycisk `OK`, aby zmiany zostały zapamiętane i zastosowane.

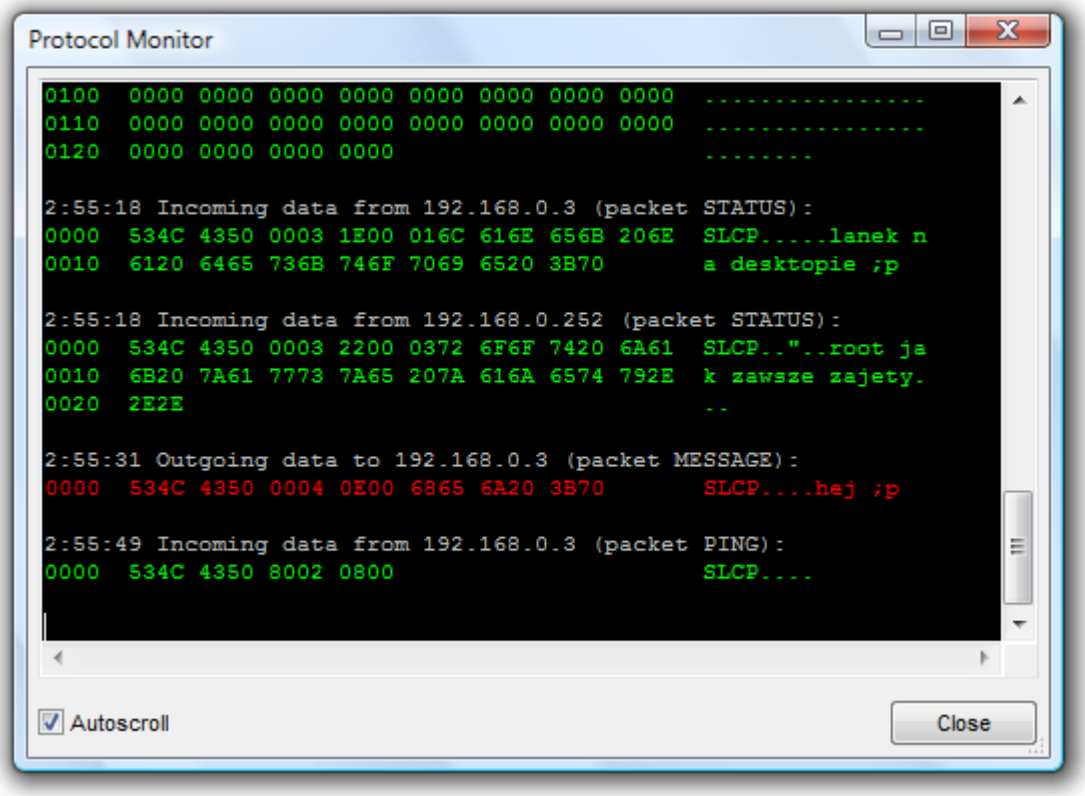
Wszystkie ustawienia zapisywane są w pliku ustawień, który można znaleźć katalogu przechowującym ustawienia i pliki konfiguracyjne innych aplikacji, zgodnie z polityką systemu, więcej o miejscu przeznaczenia można znaleźć w dokumentacji wxWidgets w sekcji dotyczącej klasy `wxStandardPaths`.

Dialog ustawień automatycznie pokaże się również w podczas pierwszego uruchomienia aplikacji oraz w przypadku kiedy nie zostanie znaleziony plik konfiguracyjny.

6.3.5. Monitor protokołu

W programie można znaleźć ciekawe i bardzo przydatne narzędzie – „Monitor protokołu". Monitor pozwala obserwować wysyłane i odbierane przez program pakiety protokołu SLCP. Zatem w łatwy sposób można śledzić generowany przez aplikację ruch sieciowy.

Narzędzie to można uruchomić z menu `Lanek->Protocol monitor`.



Rys. 6.6. Monitor protokołu

Widok okna monitora przedstawia rysunek 6.6. Jak widać, w dosyć przystępny sposób prezentowane są dane jakie „wędrują" pomiędzy aplikacjami Lanek uruchomionymi w sieci.

Narzędzie tego typu jest bardzo przydatne nie tylko w celach edukacyjnych, ale także testowych, nie trzeba uruchamiać żadnych innych dodatkowych narzędzi w postaci snifferów, aby sprawdzić co się dzieje w sieci.

Należy jeszcze wspomnieć o tym, iż proces „logowania" ruchu pakietów następuje dopiero po uruchomieniu monitora, a zostaje wstrzymany po jego wyłączeniu. W związku z tym nie jest zapamiętywany zaobserwowany ruch sieciowy, dlatego przy każdym uruchomieniu monitora okno będzie puste, aż do czasu „nadejścia" nowych danych.

6.4. Możliwości rozwoju

Jak zostało napisane we wstępie, program powstał w celach testowych i prezentacyjnych. Mimo to jest użyteczną aplikacją, choć posiada ograniczone możliwości i funkcje, a autor nie przewiduje dalszego rozwoju aplikacji w przyszłości.

Pełny kod źródłowy programu zawarty na płycie CD dołączonej do pracy rozpowszechniany jest na zasadach licencji GPL, zatem nic nie stoi na przeszkodzie, aby program nadal był rozwijany przez zainteresowane osoby lub samych użytkowników.

Do ciekawych i użytecznych funkcji o jakie można wzbogacić program, można zaliczyć:

- Rozmowy grupowe, czyli prowadzenie pogawędek w większym gronie, czy to w ściśle wybranej liczbie użytkowników, czy z cała siecią.
- Audio/video konferencje, przy których niekoniecznie trzeba się posiłkować jakimś dedykowanymi protokołami jak RTP [22].
- Przesyłanie plików pomiędzy użytkownikami.
- Zakładki okna rozmowy, grupujące wszystkie rozmowy w jednym oknie.

Autor byłbym niezmiernie wdzięczny za wszelkiego rodzaju informacje o modyfikacjach i próbach dalszego rozwoju aplikacji, które można byłoby umieścić na stronie internetowej autora [4], gdzie zapewne zostanie opublikowana niniejsza praca lub sam program.

[4] `http://malcom.pl`

7. Podsumowanie

Jak wspomniano w niniejszej pracy, standardowe protokoły transportowe takie jak TCP i UDP nie są idealne w zastosowaniach lokalnych, brakuje bezpiecznego protokołu komunikacyjnego działającego w sieciach LAN i pojedynczych segmentach sieci. Dlatego jednym z głównych celów niniejszej pracy było wypełnienie tej luki przez zaprojektowanie takiego protokołu.

Nacisk na bezpieczeństwo oraz inne cele jakie przedstawiono w pracy w dużej mierze miały wpływ na kształt powstałego protokołu SLCP. Zaprezentowane cele i założenia były odzwierciedleniem moich wizji bezpiecznego i elastycznego protokołu komunikacyjnego przeznaczonego do sieci lokalnych.

Na szczególną uwagę zasługuje sposób przedstawienia całego procesu projektowego. Poruszane kwestie i aspekty projektowe, argumentacja wyborów odpowiednich rozwiązań oraz analizy są cennym materiałem. Uważam, że proces ten był wielką, pouczającą i fascynującą lekcją.

Wybór protokołu UDP jako fundament SLCP uważam za bardzo dobre posuniecie, które rozwiązuje wiele problematycznych aspektów. Dzięki niemu nie trzeba ręcznie implementować pewnych mechanizmów sieciowych, a bazując na stosie TCP/IP, dostępnym na większości obecnych dziś architekturach i systemach, otworzyliśmy sobie długą i szeroką drogę do wielu ciekawych zastosowań protokołu SLCP.

Biblioteka slcp implementująca zaprojektowany protokół staje się bardzo ważnym narzędziem mogącym stać się jednym z głównych podpór przyszłych zastosowań protokołu. Dzięki jej elastycznej budowie, systemowi zdarzeń opartemu na wzorcu obserwatora oraz modelu obiektowym, biblioteka ta staje się bardzo łatwym, a zarazem decydującym elementem w procesie rozszerzania istniejącego oprogramowania o możliwości komunikacyjne oferowane przez protokół SLCP, a także tworzenia i projektowania nowych ciekawych usług i aplikacji sieciowych.

Łatwa rozszerzalność i modyfikacja biblioteki oraz samego protokołu mają bardzo wielkie znaczenie w wyborze rozwiązań dla przyszłych projektów wymagających bezpiecznej komunikacji lokalnej.

Mam również nadzieję, że zaprezentowany program Lanek, znajdzie zastosowanie w pro-

stej komunikacji tekstowej miedzy użytkownikami sieci lokalnej, gdzie nie ma możliwości skorzystania z innych mechanizmów. A prezentowane przykładowe scenariusze użycia nie będą jedynymi realnymi zastosowaniami protokołu SLCP.

Wszystkie cele niniejszej pracy zostały zrealizowane, a poznane w związku z tym materiały i zdobyte doświadczenie są bardzo cennym nabytkiem.

Myślę, że przedstawione materiały i analizy są idealnym źródłem i punktem wyjścia skłaniającym do dalszych analiz i testów oraz dają szerokie spojrzenie na wszystkie ważne aspekty, związane z projektowaniem protokołu oraz osiągnięciem zamierzonego celu jakim jest bezpieczna i prosta komunikacja w sieciach LAN.

Bibliografia

[1] R. Scrimger and P. LaSalle. *TCP/IP. Biblia.* Helion, 2002.

[2] J. Reynolds and J. Postel. *RFC 1700: Assigned Numbers*, October 1994.

[3] J. Postel. *RFC 793, Transmission Control Protocol*, September 1981.

[4] J. Postel. *RFC 768: User Datagram Protocol*, 28 August 1980.

[5] J. Postel (ed.). *RFC 791: Internet Protocol*, September 1981.

[6] J. Mogul. *RFC 919: Broadcasting Internet Datagrams*, October 1994.

[7] S. Deering. *RFC 1112: Host Extensions for IP Multicasting*, August 1989.

[8] *RFC 3376: Internet Group Management Protocol, Version 3*, October 2002.

[9] *Technika informatyczna. Wytyczne do zarządzania bezpieczeństwem systemów informatycznych. Pojęcia i modele bezpieczeństwa systemów informatycznych.* Polski Komitet Normalizacyjny, 1999. ISO/IEC TR 13335-1 (PN-I-13335-1:1999).

[10] G. Huston. *RFC 2990: Next Steps for the IP QoS Architecture*, November 2000.

[11] M. Degermark, B. Nordgren, and S. Pink. *RFC 2507: IP Header Compression*, February 1999.

[12] M. Degermark (ed.). *RFC 3096: Requirements for robust IP/UDP/RTP header compression*, July 2001.

[13] S. Kent and R. Atkinson. *RFC 2402: IP Authentication Header*, November 1998.

[14] S. Kent and R. Atkinson. *RFC 2406: IP Encapsulating Security Payload (ESP)*, November 1998.

[15] A. D. Birrell and B. J. Nelson. *Implementing Remote Procedure Calls.* XEROX CSL-83-7, October 1983.

[16] S. Kent and R. Atkinson. *RFC 1057: Remote Procedure Call, Version 2*, June 1988.

[17] D. Harrington and et al. *RFC 3411: An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks*, December 2002.

[18] R. Presuhn and et al. *RFC 3418: Management Information Base (MIB) for the imple Network Management Protocol (SNMP*, July 2003.

[19] B. Stroustrup. *Projektowanie i rozwój języka C++*. WNT Warszawa, 1996.

[20] J. Postel. *RFC 862: Echo Protocol*, May 1983.

[21] J. Postel. *RFC 347: Echo Process*, 30 May 1972.

[22] *RFC 3550: RTP: A Transport Protocol for Real-Time Applications*, July 2003.

Spis rysunków

Zawartość płyty CD-ROM

Wraz z niniejszą pracą dołączona jest płyta CD zawierająca:

1. Elektroniczną wersję pracy w formacie LaTeX i PDF.
2. Źródła biblioteki slcp.
3. Źródła programu Lanek.
4. Wersję binarną (Windows i Linux) programu Lanek.
5. Microsoft Visual C++ 2008 SP1 Redistributable Package (x86).