Maciej Nowak
Wydział Informatyki i Zarządzania
Wyższej Szkoły Informatyki w Łodzi

Promotor: dr Wojciech Horzelski

ROZSZERZENIE MECHANIZMÓW FUNKCJONALNOŚCI PROTOKOŁU TFTP

Streszczenie – Artykuł opisuje wdrożenie rozszerzonych mechanizmów funkcjonalności protokołu TFTP. W głównej mierze mają być to mechanizmy komunikacji pomiędzy serwerem a klientami. Pozostałe rozszerzone mechanizmy są drugorzędne i będą tylko uzupełnieniem mechanizmów funkcjonalności komunikacji.

1 Wstęp

Gdyby zapytać większość z nas o sposoby wymiany plików padłyby odpowiedzi o udostępnianiu zdjęć na serwisach społecznościowych, filmów na YouTube, czy wymiany plików przez komunikatory internetowe. Wspomniane byłyby pewnie hostingi plików oraz torrenty, lub inne systemy wymiany plików peer-to-peer, chociażby peer2mail. Część osób z pewnością wspomniała by o aplikacjach FTP. Niewiele osób wymieniłaby protokół TFTP, który ma w nazwie wymianę plików. Działoby się tak dlatego, że sam protokół TFTP jest bardzo ograniczony ze względu na jego funkcjonalność i nie nadaje się do większości zastosowań powyższych typów, dlatego używany jest do konkretnych i wąskich specjalności. Stąd też brak jego powszechności w masowych rozwiązaniach komunikacji sieciowej w Internecie.

Gdyby jednak spróbować rozszerzyć jego możliwości, aby TFTP mógł się nazywać kompletnym i funkcjonalnym rozwiązaniem w dziedzinie przekazywania plików to jakie mechanizmy musiałby spełniać? Tak, aby wymiana danych następowała w sposób masowy oraz aplikacje wymiany plików były intuicyjne dla użytkownika końcowego. Temu właśnie będzie poświęcona niniejsza praca.

Ponieważ, ze względu na efekt końcowy, standardowe aplikacje TFTP spełniają tylko dwie funkcje: wysyłania pliku oraz jego pobierania to do rozszerzenia jego funkcjonalności potrzeba:

- Wprowadzić mechanizm wyświetlania plików i katalogów w bieżącej lokalizacji.

- Umożliwić „poruszanie się” po katalogach – przemieszczanie się pomiędzy drzewem plików.
- Dać możliwość tworzenia katalogów oraz usuwania plików i katalogów.
- Wdrożyć możliwość zmiany nazw plików oraz ich przenoszenie.
- Uściślić wyświetlanie informacji o plikach o pewne atrybuty (takie jak rozmiar, czas utworzenia plików, atrybuty uprawnień czytania, pisania bądź wykonania).
- Wprowadzić pewne mechanizmy autoryzacji i określonych uprawnień do wymiany plików.

Aby osiągnąć te efekty niezbędne jest wprowadzenie dodatkowych struktur pakietów oraz uzupełnienie tych już istniejących o inne opcje. W teorii wiązać będzie się to ze zdefiniowaniem pakietów oraz prezentacją ich wymiany. W implementacji będą to praktyczne realizacje komunikacji.

2 Protokół TFTP – podstawowe informacje

Posiadając informację co do funkcji budowy serwera oraz klienta UDP trzeba zastosować je do systemu TFTP. Warto tutaj przypomnieć, że implementacja będzie w zasadzie opierała się w większości o specyfikację RFC. Skoro tak to warto pokrótce omówić te funkcje.

PAKIET RRQ/WRQ

2 bajty	napis	1 bajt	napis	1 bajt
01/02	Nazwa pliku	0	Tryb	0

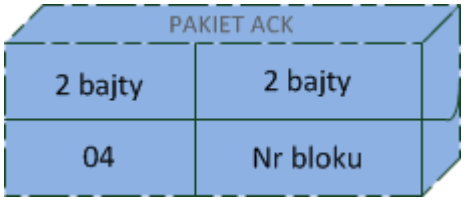
Rys. 1. Komunikat RRQ,WRQ

Klient wysyłając do serwera żądanie czytania pliku (**RRQ** – read request) przekazuje komunikat. Pierwsze 2 bajty przeznaczone są na nr komunikatu (optcode) . Następne pole to nazwa pliku zakończone bajtem zerowym, na końcu podawany jest tryb pliku, w którym reprezentowany jest sposób przesyłania tekstu. Tryb *netascii* wskazuje, że dane są wierszami tekstu ASCII zakończonymi sekwencją powrotu karetki (znaków CR/LF). W takim przypadku zarówno klient jak i serwer muszą dokonywać konwersji pomiędzy takim formatem a znakami. Drugim możliwym trybem jest *octet*. W tym przypadku nie potrzeba żadnej konwersji – znaki przekazywane są jako 8-bitowe bajty. Komunikat taki jest zakończony bajtem 0. Komunikat **WRQ** różni się od

RRQ tylko nr opcode i służy do żądania zapisu pliku. Po prawidłowym otrzymaniu takiego komunikatu serwer powinien sprawdzić, czy zapytanie jakie dostał jest możliwe do zrealizowania. Jeśli tak wysyłany jest komunikat potwierdzający ACK. Każdy taki pakiet zawiera nr bloku, który służy potwierdzeniu odbioru. Potwierdzenie czytania lub zapisu pliku uzupełnione jest bajtem 0. Gdy serwer wysyła dane (po otrzymaniu pakietu RRQ) przesyła pakiet **DATA**. Dane są dzielone na poszczególne fragmenty tak, aby mogły się zmieścić w rozmiarze komunikatu. Dane umieszczane są po polu opcode oraz po nr bloku. Każdy kolejny nr bloku jest inkrementowany. Jeśli wysyłany jest ostatni pakiet DATA doklejany jest bajt 0 na jego końcu. Po każdym takim wysłanym pakiecie serwer czeka na potwierdzenie ACK (tzw. stop-and-wait protokół). Po zapytaniu RRQ lub WRQ nie musi następować potwierdzenie ACK. Może zdarzyć się, że serwer z jakichś powodów nie może wysłać lub zapisać pliku, wysyła w takim przypadku komunikat **ERROR**. Błędy takie mogą się zdarzyć również w trakcie przesyłania pliku (zapisu lub odczytu), dlatego pakiet ERROR oprócz standardowego dla wszystkich komunikatów pola opcode, zawiera nr błędu oraz opis tego błędu. Oprócz problemów związanych z plikami mogą również występować problemy sieci (zagubienie lub powielenie pakietu). Implementacja systemu powinna zawierać algorytmy odmierzania czasu oraz retransmisji.

Rys. 2. Komunikat DATA

Nadmienić również trzeba, że pierwsza specyfikacja RFC (1350) określa maksymalny rozmiar pakietu danych jaki powinien być wysyłany na 512 bajtów. Został on nałożony w związku z koniecznością fragmentacji



Rys. 3. Komunikat ACK

Nadmienić również trzeba, że pierwsza specyfikacja RFC (1350) określa maksymalny rozmiar pakietu danych jaki powinien być wysyłany na 512 bajtów. Został on nałożony w związku z koniecznością fragmentacji datagramów. Ponieważ datagramy i tak wysyłane są pojedynczo, więc kolejna fragmentacja dodatkowo wpływa na szybkość ich przekazywania. Specyfikacja również definiuje domyślny nr portu serwera TFTP (69).

PAKIET ERROR

2 bajty	2 bajty	napis	1 bajt
05	Nr błedu	Opis błedu	0

Rys. 4. Komunikat ERROR

Kolejne specyfikacje rozszerzyły nieco możliwości protokołu TFTP. W pierwotnej wersji tych komunikatów jest 5, w kolejnych już 6 powiększonych o dodatkowe opcje i tak kolejne dokumenty wprowadzają nowy komunikat **OACK**. Służy on tylko do potwierdzenia RRQ lub WRQ wraz z opcjami.

PAKIET OACK

2 bajty	napis	1 bajt	napis	1 bajt	napis	1 bajt	napis	1 bajt
06	Opcja1	0	Wartość1	0	OpcjaN	0	WartośćN	0

Rys. 5. Komunikat OACK

PAKIET RRQ/WRQ wraz z opcjami

2 bajty	napis	1 bajt	napis	1 bajt	napis	1 bajt	napis	1 bajt	napis	1 bajt	napis	1 bajt
01/02	Nazwa pliku	0	Tryb	0	Opcja1	0	Wartość1	0	OpcjaN	0	WartośćN	0

Rys. 6. Komunikat RRQ,WRQ z opcjami

Po wysłaniu przez klienta zapytania RRQ lub WRQ z opcjami serwer odpowiada komunikatem OACK. Po otrzymaniu OACK klient potwierdza ten pakiet komunikatem ACK. Na opcje wraz z wartościami w OACK przeznaczone jest 512 bajtów. Nazwy opcji oraz wartości są znakami ASCII (RFC2347).

Kolejne specyfikacje określają co mogą zawierać pola opcji oraz jakie wartości przyjmować, m.in.:

- RFC2348 – rozmiar bloku wysyłanych danych. Domyślnie jest to wartość 512 bajtów danych i bez opcji takie wartość są potwierdzane pakietem ACK. Dokument przekazuje następujące rozmiary: 512, 1024, 1428, 2048, 4096, 8192 bajtów. Nazwa opcji to w tym przypadku: „blksize"
- RFC2090 – przesyłanie danych przy pomocy adresu multicast i również o takiej samej nazwie opcji. Wartość pola to: „adres, port, flaga"
- RFC2349 – negocjowanie przez klienta czasu po jakim następuje retransmisja („timeout") oraz wielkość transferu („tsize") wyrażoną w bajtach.

Dodatkowo RFC3617 – określa możliwość żądań za pomocą URI, dokument precyzuje dokładnie w jakiej formie taki adres powinien wyglądać.

Warto tutaj zaznaczyć, że programista może tutaj samemu implementować różnego rodzaju opcję np. określać atrybuty plików (właścicieli, prawa dostępu itp.).

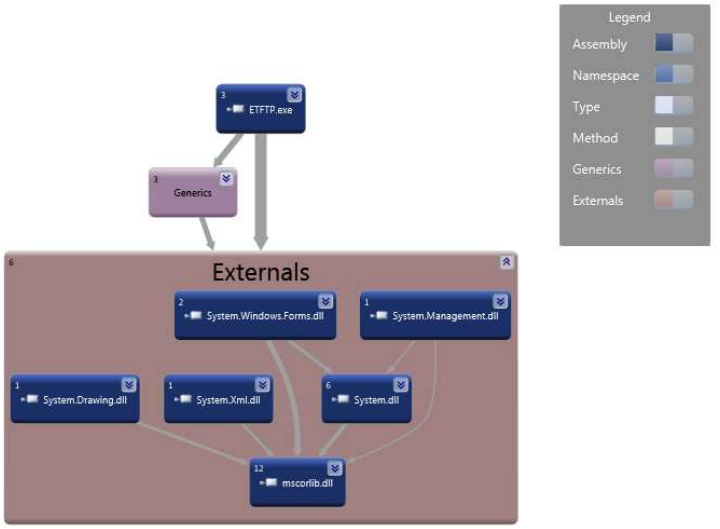
Na koniec omawiania protokołu warto zwrócić tutaj uwagę na bezpieczeństwo. Dokumenty RFC nie precyzują miejsca na umieszczanie nazwy użytkownika oraz hasła. Można to oczywiście zrobić za pomocą opcji jeśli implementacja pozwala na takie zastosowanie. Wiąże się to jednak z każdorazowym przesyłaniem otwartym tekstem loginu oraz hasła. Dlatego też w większości serwerów TFTP występuje ograniczenie co do dostępu do określonych plików. Możliwe jest również ustawienie odpowiednich uprawnień w systemach Unix poprzez nadanie odpowiedniej wartości ID użytkownikowi lub grupie.

3 Wymagania protokołu, architektury, sprzętowe i aplikacji

Na potrzeby systemu wymiany plików niezbędne jest stworzenie nowego protokołu sieciowego ETFTP (*ang. Expanded Trivial File Transfer Protocol*). Będzie to w zasadzie rozszerzenie już istniejącego i opisanego w dokumentach RFC protokołu TFTP. Protokół ten posłuży do zrealizowania rozszerzonych mechanizmów komunikacji sieciowej. ETFTP ma rozszerzać możliwości funkcjonalne aplikacji opisane w projekcie. Rozszerzenie funkcjonalności będzie polegać na definicji nowych struktur danych – pakietów, rozbudowę tych istniejących o dodatkowe opcje oraz ukazanie przepływu danych pomiędzy aplikacjami. Komunikacja w systemie wymiany plików będzie odbywać się poprzez strukturę sieciową serwer ETFTP-klienci. Trzeba założyć, że

protokół ETFTP oraz jego mechanizmy mają być niezależne od rodzajów sieci, czy architektury sieciowej i mają działać w szeroko pojętym Internecie.

Te obostrzenia nie dotyczą jednak samych aplikacji. Ponieważ do poprawnego działania systemu sieciowego niezbędne jest, żeby serwer jak i klient komunikowali się z systemem plikowym. Serwer po otrzymaniu żądania od klienta pliku musi skomunikować się przy udziale systemu operacyjnego z jądrem systemowym, aby wydzielił mu pewne zasoby. Zasoby te nie muszą być spójne z zasobami klienta. Maszyny mogą posiadać zupełnie innym system plików. Ponieważ serwer pracuje pod innym systemem plików niż klient – informacje o samych plikach (atrybuty) powinny być przesłane w opcjach i na drugiej maszynie odpowiednio zinterpretowane. Podobnie sprawa się ma z trybem odczytu kodów ASCII (zamiana znaku nowej linii) i odpowiednią interpretacją przez konkretne środowiska. Dlatego niezbędne jest zdefiniowanie jakiegoś standardu przesyłania danych niekompatybilnych.



Rys. 7. Zewnętrzne biblioteki klienta ETFTP

Żeby ułatwić proces tworzenia projektu i jego wdrażania trzeba popełnić pewne założenia.

Przede wszystkim aplikacje będą tworzone na systemach operacyjnych z architekturą 32-bitową. Serwer ETFTP to usługa pracująca na maszynie z systemem operacyjnym spod znaku UNIX. Usługa powinna pracować w środowisku konsolowym. Klient natomiast będzie pracował w trybie graficznym pod kontrolą systemu Windows.

Klient będzie stworzony przy użyciu środowiska programistycznego Microsoft Visual Studio 2010 w języku programowania c#. Do poprawnej pracy niezbędne będą biblioteki sieciowe .NET w wersji przynajmniej 4.0. Aplikacja klienta będzie kompilowana dla architektury 32-bitowej. Kod serwera natomiast będzie dostępny pod c++ i skompilowany zostanie przy użyciu g++.

Użycie środowiska graficznego w kliencie ETFTP ułatwia posługiwanie się aplikacją poprzez narzucenie prostoty i intuicyjności. Do działania serwera natomiast potrzebne jest jedynie konsolowe środowisko o niewielkich zapotrzebowanych sprzętowych.

4 Projekt systemu wymiany plików

Na wstępie warto wyróżnić minimalne niezbędne założenia dla realizacji zagadnień związanych z korzystaniem z mechanizmów komunikacji w systemie wymiany plików. Ponieważ system ten będzie w całości oparty na protokole transportowym UDP niezbędne będzie zaimplementowanie podstawowych opcji wynikających z jego ułomności. Następnie zaprezentowanie zostaną metody, które posłużą przy wdrożeniu funkcjonalności użytkowej.

Aby połączenie UDP nie było zawodne trzeba wprowadzić do niego czas oczekiwania i ponownej transmisji. Zliczać kolejne datagramy w implementacji, oraz przechowywać ostatni nie potwierdzony, aby klient wiedział którego żądania dotyczy odpowiedź. Następne wymaganie niezbędne wprowadzenia to zegar, który po określonym czasie będzie retransmitował zgubione pakiety.

W przypadku takiego systemu wymiany plików wprowadzony zostanie tzw. Liniowy licznik zegarowy (ang. Linear timer) ponownej transmisji. Oznaczać to będzie, że obsługa czasu oczekiwania i ponowienia transmisji polegać będzie na wysłaniu żądania i czekaniu przez określoną stałą liczbę sekund. Jeśli nie nadejdzie odpowiedź to żądanie będzie ponowione. Po określonych ilościach retransmisji żądanie przestaje być przetwarzane.

Jest to bardzo prosty mechanizm czasu retransmisji i nie uwzględnia odległości pomiędzy punktami końcowymi, obciążenia sieci czy jej szybkości. Na potrzeby tego systemu wymiany plików jednak w zupełności wystarczy. Można oczywiście wprowadzić zegar który niweluje te niedogodności i inne algorytmy określające RTO (ang. *Retransmision timeout)* Taki algorytmy przedstawione są w literaturze i nie będą one tutaj rozpatrywane.

Przy rozpatrywaniu czasów oczekiwania i retransmisji możemy spotkać się z trzeba możliwościami.

- Zaginęło żądanie
- Zaginęła odpowiedź

- Jednostka RTO jest zbyt mała

Pierwsze dwa wiążą się z ponowieniem żądania, trzecia możliwość ze zwielokrotnieniem żądania i odrzuceniem jednego z nich.

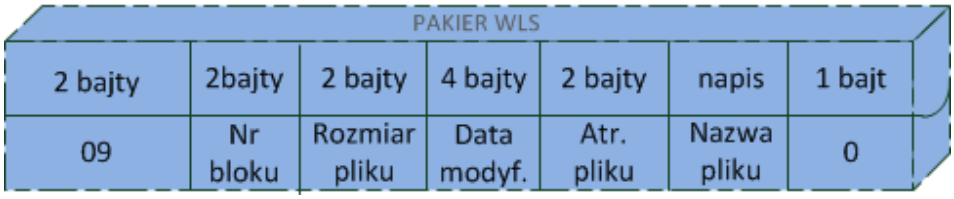
Dodatkowo trzeba przyjąć, iż czasy retransmisji oraz ilość powtórzeń powinna być uzgadniana pomiędzy stronami. Zaimplementowane powinno to być w opcjach zgodnie z RFC dla TFTP.

5 Wyświetlanie listy plików oraz informacji o nich i poruszanie się po drzewie katalogów

Wyświetlanie listy plików wiązać będzie się z dodaniem nowego komunikatu TFTP.

Rys. 8. Pakiet RLS

Pierwsze dwa bajty tak jak we wszystkich pakietach to numer opcode, dla pakietu RLS (read list) to numer 8. Następnie pole DirPath, to ścieżka względna lub bezwzględna katalogu z którego ma być czytana lista plików. Poruszanie się po katalogach może odbywać się poprzez przekazywanie względem lokalizacji zapisanej w serwerze ścieżki katalogu domyślnego lub poprzez bezwzględne odwołanie. Potwierdzenie następuje podwójną wymianą komunikatu ACK. Następnie serwer przesyła pakiet według struktury **WLS** (write list) o numerze OpCode 9.



Rys. 9. Pakiet WLS

Pakiet taki składa się jak wspomniano w pierwszych z dwóch bajtów z pola OpCode. Następne 2 bajty to inkrementowany numer

listowanego pliku. Kolejno rozmiar pliku wyrażony w bajtakch 2 bajty, data modyfikacji pliku w systemie Unix od 1970r. – 4 bajty, atrybuty pliku – 2 bajty, nazwa pliku zakończona bajtem 0.

Struktura tego pakietu w implementacji wygląda następująco:

```
struct etftphdr {
        u_int16_t th_opcode;
        u_int16_t th_block;
        u_int16_t th_size;
        u_int32_t th_date;
        u_int16_t th_mode;
        char th_filename[1];
        };
```

Potwierdzenie wysłania ostatniej informacji o pliku odbywa się również dwoma komunikatami ACK. Ścieżkę katalogu w pakiecie RLS trzeba ograniczyć do 512 bajtów. Poruszanie się po bardzo zagnieżdżonych katalogach nie będzie więc możliwe lub trzeba będzie zmienić domyślną ścieżkę katalogów zapisaną na serwerze.

W pakiecie WLS warto przyjrzeć się informacją o pliku. Data pliku to nic innego jak ilość sekund w systemie Unixowym od 01.01.1970r. Aby wyświetlić poprawnie ostatnią modyfikację pliku trzeba w kliencie odpowiednio przetworzyć te dane. Rozmiar pliku wyrażony jest w bajtach, natomiast atrybuty pliku to 9 bitów poświęcona na uprawnienia do czytania pisania bądź wykonywania, pozostałe mówią o rodzaju pliku (np. katalog, dowiązanie symboliczne itp.) oraz flagi.

Wysłanie pakietu RLS może objawić się odesłaniem przez serwer pakietu błędu. Komunikaty błędów, które mogą wystąpić to brak takiego katalogu, brak praw czytania katalogu oraz niezidentyfikowany błąd.

6 Komunikacja klienta i serwera ETFTP

Na potrzeby systemu dla ułatwienia opisu warto zdefiniować domyślne porty na których będą pracować aplikacje. Dla serwera mógłby być to port przeznaczony dla serwera TFTP 69. Jednak jeśli z jakiś powodów nie można go używać – np. jest zajęty, bądź brak uprawnień administratora dla rezerwacji portów poniżej 1024, warto określić inne porty. Dla serwera ETFTP będzie to port 3344, natomiast dla klienta 4433.

Dla poprawnego działania systemu wymiany plików warto ustawić również pewne opcje, tak aby obsługa klientów przez serwer odbywała się w sposób niezawodny. Po pierwsze przy pomocy ustawień gniazda i opcji SO_RCVBUF można powiększyć rozmiar bufora gniazda. Przy

wywołaniu recvfrom datagram jest przekazywany z tego bufora do przetwarzania zgodnie z porządkiem FIFO. Gdy do gniazda przybyło zbyt wiele datagramów, zanim proces zdążył je obsłużyć tworzyła się kolejka. Przepełnienie takiej kolejki wiąże się ze zgubieniem datagramów. Określić należy, więc rozmiar bufora na jeden z największych dozwolonych 480 pakietów po 512 bajtów. Oznaczać to będzie również, że liczba potencjalnych jednoczesnych klientów nie powinna przekraczać tej liczby na jedno gniazdo.

Na stacji, gdzie jest tylko jeden interfejs oraz jeden adres IP serwer nie musi martwić się o odpowiednie dowiązanie nowej otrzymanej struktury adresowej przez funkcję recvfrom. Problem zaczyna się gdy tych interfejsów jest więcej. Nie wiadomo na który interfejs przyjdzie odpowiedź i czy ten źródłowy jest potencjalnym adresem docelowym do którego mają płynąć odpowiedzi. Dowiązanie pojedynczego gniazda może sprawić problem, gdy serwer nie posiada żadnych informacji o interfejsach. Aby tego uniknąć, w implementacji klienta można sprawdzić nazwę domenową stacji serwera. Drugim rozwiązaniem jest wywołać funkcję select - serwer czeka na gotowość gniazda i konkretnego interfejsu, aż stanie się dostępne dla czytania, a następnie wysyła odpowiedź z tego gniazda, które jest gotowe.

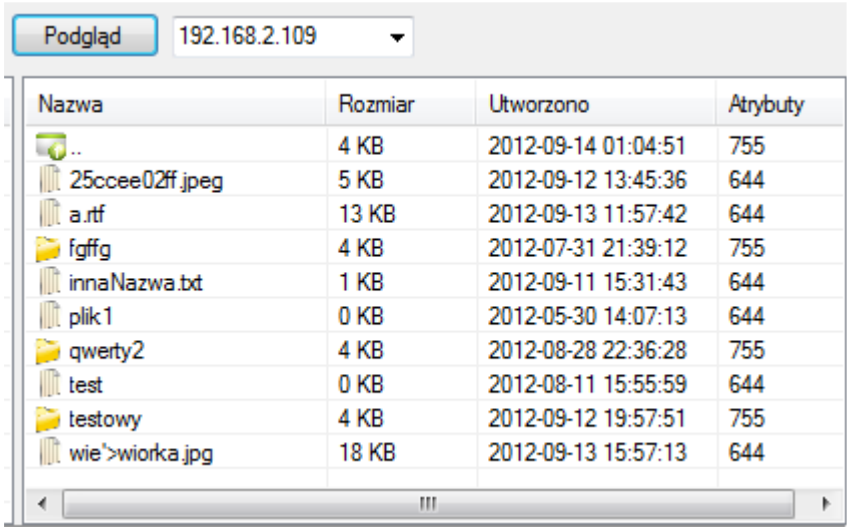
Do monitorowania wszystkich interfejsów można użyć funkcji get_ifi_info, aby natomiast określić interfejs wyjściowy trzeba użyć opcji gniazd, IP_RECVIF lub IP_PKTINFO.

7 Rozszerzone mechanizmy komunikacji

Warto zwrócić uwagę jak dla rozszerzonych mechanizmów komunikacji wygląda przepływ sterowania oraz informacji.

Dla wyświetlenia zdalnej listy plików na serwerze klient realizuje żądanie RLS. W kliencie wywołana zostaje metoda bListFiles, która inicjuję metodę pomocniczą ListRemoteFiles, w niej to tworzony jest nowy obiekt typu EtftpPackage, w którym przygotowany jest pakiet z numerem opcode 08 oraz ścieżką do zdalnego katalogu. Tak przygotowany pakiet jest wysyłany poprzez utworzone gniazdo udp i wcześniej zdefiniowane ustawienia sieciowe na dobrze znany nr portu serwera. Gdy pakiet trafia na interfejs sieciowy, który jest dowiązany do serwera etftp, oraz odpowiedni adres sieciowy (adres IP oraz port) serwer wywołuje nowy wątek. Sterowanie zostaje przekazane do Menedżera listy. Pakiet jest interpretowany przez metody klasy PackageManager. Po sprawdzeniu poprawności pobierane są informację odnośnie żądania z systemem plików. Gdy jest możliwe zrealizowanie żądania (prawidłowo otwarty zostanie katalog do odczytu) wywoływana zostaje metoda do wysłania pakietu ACK. Pakiet wysyłany jest na adres IP i port źródłowy skąd przyszło żądanie z portu

efemerycznego. Klient po wysłaniu żądania przechodzi w stan nasłuchiwania odpowiedzi, gdy otrzyma on datagram sprawdzana zostaje możliwość wyświetlenia listy plików. Gdy wszystko jest w porządku zwracany jest komunikat ACK na port, z którego przyszła odpowiedź. Potwierdzenie ACK jest dalej realizowane przez ten sam wątek, który został wywołany na początku żądania, ponieważ skierowany został do portu utworzonego w tym wątku i nasłuchującego właśnie na tym porcie. Po przetworzeniu pakietu czytany jest katalog: nazwa pliku, rozmiar, data utworzenia oraz atrybuty dostępu do pliku. Gdy wszystkie informacje zostają uzyskane zostają one "zlepione" w odpowiednią strukturę uzupełnioną o nr pliku i pole opcode oraz przesłane do klienta przy pomocy komunikatu WLS. Gdy do klienta trafił odpowiedni co do kolejności plik potwierdzenie następuje standardowym komunikatem, wtedy serwer wysyła informacje o następnym pliku. Procedura się powtarza, aż końca listy plików do czytania z katalogu. Ostatnie potwierdzenie wysłane przez klienta jest potwierdzane komunikatem ACK.



Rys. 10. Możliwa realizacja wyświetlenia listy plików w oknie klienta ETFTP

Taki przesłanie komunikatów wiąże się z pojawieniem listy plików oraz informacji o nich w oknie interfejsu graficznego klienta. Przykład takiego wyświetlenia można zaobserwować na rysunku 10. W oknie widać listę plików i katalogów oznaczonych odpowiednimi ikonami, ich nazwy, rozmiar, atrybuty oraz datę utworzenia. Informacje o tym czy plik jest katalogiem pobierana jest z jego atrybutów. Dodatkowo katalog z dwoma kropkami umożliwia przejście w dół w ścieżce katalogów.

Realizacja takiego scenariusza wymiany komunikatów jest idealna. Lokalnie ciężko o zagubienie się pakietu oraz o tak duże opóźnienia, aby komunikacja stała się uciążliwa. W środowisku rozproszonym trzeba jednak zadbać o wdrożenie mechanizmów retransmisji w przypadku zagubienia pakietu i zbytnich opóźnień. Realizacja tych funkcji nie wykracza jednak poza wdrożenie ich z protokołu TFTP.

W podobny sposób jak wyświetlanie plików użytkownik może się „poruszać" po drzewie katalogów. W oknie wyświetlanych plików kliknięcie katalogu wiąże się z uzupełnieniem ścieżki o dodatkową nazwę, natomiast zejście w dół drzewa katalogu to wysłanie do serwera ścieżki z doklejonymi dwoma kropkami.

Podobnie realizowane są pozostałe żądania rozszerzonych mechanizmów komunikacji. Nie są one aż tak rozległe jak w przypadku komunikatu wyświetlenia listy plików. Żądania usunięcia, przeniesienia, zmiany nazwy, czy utworzenia plików wiąże się tylko z jedną odpowiedzią: potwierdzeniem lub błędem, a mechanizm przekazywania sterowania jest identyczny. Zmieniają się za to metody wywoływane w poszczególnych klasach.

8 Podsumowanie

Celem pracy było zrealizowanie tematu rozszerzenia mechanizmów funkcjonalności protokołu TFTP. Realizacja takiego zadania opierała się na zdefiniowaniu pojęć, określeniu podstaw, w jaki sposób projekt ma zostać wykonany, narzuceniu pewnych wymagań oraz ograniczeń środowiska, wytyczeniu założeń projektowych systemu wymiany plików, zaimplementowaniu tych założeń i przetestowaniu gotowej usługi. Do końcowych zadań tematyki pracy należało sporządzenie dokumentacji dla aplikacji tworzącej system wymiany plików oraz przedstawienie wniosków i podsumowanie realizacji zadania. Rozdział ten będzie przeznaczony na opis ostatniego elementu zaprezentowanego planu.

Ponieważ temat pracy zakładał odwołanie się do protokołu TFTP, niezbędnym było zdefiniowanie podstawowych pojęć obejmujących pewien zakres tych zagadnień. Określone zostały więc definicje protokołu TFTP, UDP oraz IP. Definicje te były ukierunkowane na realizację tematyki, podobnie jak definicje pewnego zbioru funkcji (gniazda UDP), które posłużyły realizacji praktycznej projektu.

Opierając się na TFTP niezbędnym stało się sięganie do dokumentów RFC, tak aby projekt był możliwie zgodny z jej specyfikacją, jeśli chodzi o podstawy. Te podstawy opisane w rozdziale 2 stanowią właściwą wykładnię pod budowę założeń systemu i protokołu ETFTP.

Zanim jednak zrealizowany został projekt potrzeba było określić wymagania systemu co do samego protokołu, architektury sieci, sprzętu

i aplikacji. Zostało to przedstawione w rozdziale 3 poświęconym zbiorze wymagań systemu oraz jego obostrzeń. W rozdziale tym znalazło się również miejsce na narzędzia informatyczne, które były używane podczas tworzenia projektu, w szczególności jego implementacji. Z ważnych wymagań odnośnie kwestii sieciowej oraz oprogramowania, to oparcie projektu o rozległą sieć Internet i zrealizowanie aplikacji typu klient-serwer, gdzie klientem jest program oparty o system Windows, natomiast serwerem usługa pracująca na systemie Unix.

Rozdział 4 posłużył do zdefiniowania funkcjonalnego modelu systemu oraz zbudowania modelu logicznego aplikacji. W szczególności ukazane zostały interakcję pomiędzy klientami a serwerem. Pokazana została struktura aplikacji, pakietów oraz przepływ danych w aplikacji jak i w systemie. Uwidoczniony został również schemat sterowania pomiędzy elementami aplikacji. To w tym rozdziale zdefiniowany został nowy protokół ETFTP oraz powstały założenia projektowe co do budowy aplikacji opartej właśnie o ten protokół.

Tworzenie systemu wymiany plików od strony implementacji opisane zostało w rozdziale 5, gdzie została zaprezentowana fizyczna aplikacja klienta oraz serwera. Ten rozdział składa się przede wszystkim z kodu źródłowego z licznymi komentarzami, które ukazują działanie aplikacji, a w szczególności przepływ danych pomiędzy nimi. W tym rozdziale poświęcone zostało również miejsce na testowanie systemu i analizę wymiany komunikatów oraz monitorowanie w jaki sposób aplikacje zachowują się w dłuższej perspektywie działania. Niezbędnym było w tym miejscu pokuszenie się o opis pewnych cech całego systemu, które zaprezentują jego wady oraz zalety. Znalazły się również informacje o licznych błędach w systemie.

W rozdziale 6 zaprezentowana została dokumentacja przygotowana dla użytkownika oraz administratora. W dokumentacji dla klienta (użytkownika) zaprezentowane zostały przede wszystkimi sposoby obsługi aplikacji, wyszczególnienie funkcjonalności oraz sposoby realizacji tej funkcjonalności. Dodatkowo można znaleźć informację o instalacji oraz uaktualnieniach aplikacji. Do dyspozycji administratora udostępniona została pomoc, w której oprócz funkcjonalności administrator może się dowiedzieć o sposobach zarządzania serwerem jego ustawieniach oraz dostępnych opcjach. Do dokumentacji dołączona została również lista błędów (ang. *BugFix*) oraz proponowanych zmian (ang. *ChangeLog*) i rozszerzeń aplikacji (ang. *ToDo*).

W ramach pracy zrealizowany został projekt prostego systemu wymiany plików. Do systemu dodana została taka funkcjonalność, która realizuje cele przedstawione w pierwszym rozdziale. Opis tych dodatkowych możliwości stanowi nowo zdefiniowany protokół ETFTP, natomiast realizacja rozszerzonego projektu wymiany plików stanowi

usługę ETFTP. Przyglądając się celom z perspektywy ukończonego projektu można pokusić się o wnioski z nich płynące.

Jako cele w głównej mierze ukazane zostały te przykłady rozszerzające mechanizmy funkcjonalności protokołu. Mianowicie wyświetlanie plików i katalogów odbywa się w kliencie, jako realizacja jego żądań w sensie sieciowym. Fizycznie ukazywana jest nazwa pliku oraz jego ikona dotycząca jego typu (folder, plik) w interfejsie graficznym. Dodatkowo, można by było rozszerzyć i tą funkcjonalność o dodatkowe typy z uwzględnieniem rozszerzeń plików i prezentacją w odpowiedniej formie. Na przykład dla pliku mp3 byłaby to ikona związana z nutą. Nie zostało to jednak zrealizowane, ponieważ dla mechanizmów komunikacyjnych nie stanowi to żadnej różnicy.

Gdy już wyświetliliśmy jakąś listę plików z danego katalogu, to wypadałoby mieć możliwość zmiany lokalizacji. Jest to drugi cel rozszerzający, który został wyszczególniony, czyli możliwość przemieszczania się pomiędzy drzewem katalogów. Pod względem komunikacji stanowi pewne wyzwanie, ponieważ znacząco rozszerza funkcjonalność jak i implementację systemu. Do zrealizowania tego celu niezbędne jest wprowadzenie dwóch nowych struktur komunikatów i wdrożenie algorytmu ich przenoszenia. Od strony aplikacji natomiast niezbędne jest stworzenie drzewa katalogów w zależności od względnej lub bezwzględnej ścieżki plików. Realizacja takiej implementacji w ten sposób znacząco rozbudowuje prosty protokół wymiany plików. Gdy ilość plików w katalogu znacząco wzrasta, rośnie również ilość komunikatów, oraz czas wyświetlenia wszystkich. Jeśli do tego dodamy opóźnienia może się okazać, że jest on zbyt długi na wyświetlenie ich wszystkich.

Dodatkowo, żeby zwiększyć trochę rozmiar komunikatu poza tylko nazwą pliku, dodane zostały do komunikatu inne pola, które traktują o statystycznych informacjach o pliku. (takie jak rozmiar, czas utworzenia plików, atrybuty uprawnień czytania, pisania bądź wykonania). Mając na uwadze różne oprogramowanie systemowe u klienta oraz serwera i tym samym różny system plików, a w szczególności różną charakterystykę uprawnień do nich, niezbędna stała się konwersja atrybutów uprawnień. Taka konwersja rodzi pewne obawy, że nie można bezkarnie migrować aplikacjami pomiędzy różnymi systemami i jest ona uzasadniona. Należy ściśle przestrzegać ograniczeń systemowych. Co jest jedną z wad tego sposobu komunikacji.

Co do następnych celów, to są one ze sobą powiązane i spełniają rolę podobną jak w protokole FTP. Jest to możliwość tworzenia katalogów oraz usuwania plików i katalogów oraz zmiany nazw plików oraz ich przenoszenie. O ile w przypadku protokołu FTP występuje znaczący przyrost nadmiarowych komunikatów (meta danych), o tyle w przypadku tego systemu, komunikaty te wymieniane są jednorazowym

żądaniem oraz odpowiedzią, co jest zdecydowaną zaletą, zwłaszcza, gdy jest dużo operacji tego typu. Pewną niedogodnością mogą być proste mechanizmy autoryzacji, przy próbie operacji na tych plikach.

W pracy ostatnim celem jest wprowadzenie mechanizmów autoryzacji i określonych uprawnień do wymiany plików. Nie udało się jednak tego zrealizować programowo, a jedynie systemowo. Wyjściem jest zawężenie uprawnień pliku wykonywalnego do jakiejś grupy użytkowników i sformułowanie odpowiednich praw oraz umieszczenie pliku w hermetycznym katalogu z dostępem do określonych katalogów. Zadanie to jednak zostało przerzucone z programisty na administratora, dlatego można uważać to za sporą wadę tego systemu pod względem bezpieczeństwa. Brak uwierzytelniania oraz szyfrowania danych dla protokołu TFTP, sprawił, że jest on opcjonalnie dostępny w systemie Windows Vista, a w nowszych wersjach systemu operacyjnego jest w ogóle niedostępny.

Na wstępie padło pytanie, czy można w taki sposób rozszerzyć TFTP, aby był on funkcjonalny, powszechny i popularny. Posiłkując się niniejszą pracą nie można na to pytanie jednoznacznie odpowiedzieć. Należy przede wszystkich wziąć pod uwagę, że protokół ETFTP powstał na protokole UDP z myślą o jego prostocie i szybkości działania. Funkcjonalność, którą realizuje nadaje się do małych plików i do dużej ilości oraz różnych realizacji zadań. Jeśli doda się do niego szyfrowanie oraz autoryzację, wydłuży to znacznie czas tych operacji. Rozwiązaniem mogłoby być wprowadzenie pewnego opakowania dla TFTP, ale to już nie stanowiłoby jego rozszerzenia, a raczej bazę dla TFTP. Ponieważ aplikacja się wciąż rozwija autor ma nadzieję na poszerzenie funkcjonalności.

Literatura

[1] Comer D. E., *Sieci komputerowe TCP/IP, zasady, protokoły, architektura i architektura,* Tom I, Wydanie II

[2] CodeIt Right, URL: http://submain.com/products/codeit.right.aspx [22.06.2012]

[3] Debugger DDD, URL: http://www.gnu.org/software/ddd/ [22.06.2012]

[4] Dokument RFC 1350, URL: http://tools.ietf.org/html/rfc1350 [30.03.2012]

[5] EA, URL: http://www.sparxsystems.com/ [22.06.2012]

[6] Gedit edytor tekstowy, URL: http://projects.gnome.org/gedit/ [22.06.2012]

[7] Lista narzędzi UML, URL: http://pl.wikipedia.org/wiki/Lista_narzędzi_UML [22.03.2012]

[8] Microsoft Visio, URL: http://office.microsoft.com/pl-pl/visio/ [22.06.2012]

[9] Microsoft VS, URL: http://www.microsoft.com/poland/developer/ [22.06.2012]

[10] Testowanie oprogramowania, URL: http://pl.wikipedia.org/wiki/Testowanie_oprogramowania [30.03.2012].

[11] TFTP, URL: http://en.wikipedia.org/wiki/Trivial_File_Transfer_Protocol [30.03.2012]

[12] TFTP, URL: http://pl.wikipedia.org/wiki/Trivial_File_Transfer_Protocol [30.03.2012].

[13] Pakiet kompilatorów języka c/c++, URL: *http://gcc.gnu.org/* [22.06.2012]

[14] Polecenie Make, URL: *http://pl.wikipedia.org/wiki/Make* [30.03.2012]

[15] Rochkind, Marc J., *Programowanie w systemie UNIX dla zaawansowanych*, Wydanie trzecie zmienione i rozszerzone, Warszawa 2007, ISBN: 978-83-204-3253-4

[16] Serwis społecznościowy, URL: http://pl.wikipedia.org/wiki/Serwis_społecznościowy [30.03.2012]

[17] Stevens, W. Richard, *Biblia TCP/IP* Tom I. Protokoły, s. 245 - 260, Warszawa 1998,

[18] Stevens, W. Richard, *UNIX Programowanie Usług Sieciowych* Tom I, Wydanie II,

[19] VM VirtualBox, RUL: https://www.virtualbox.org/ [22.06.2012]

[20] Wireshark, *Sniffer sieciowy*, URL: http://www.wireshark.org/ [22.06.2012]

THE EXTENSION FUNCTIONALITY OF TFTP PROTOCOL

Summary – The article describes the implementation of the extended TFTP functional mechanisms. Largely to be a communication mechanisms between the server and the clients. Other mechanisms are secondary and only extended will complement the functional mechanisms of communication.