Protokół TCP — motywacje

- zawodne przenoszenie może powodować gubienie, niszczenie, duplikowanie pakietów lub dostarczanie ich w złej kolejności
- programy użytkowe często przesyłają dane o dużej objętości — zawodne dostarczanie bez użycia połączenia wymaga od programistów skonsturowanie mechanizmu wykrywania i obsługi błędów transmisji
- potrzeba ogólnego rozwiązania problemów zwiazanych z przesyłaniem danych niezawodnymi strumieniami, z którego korzystałyby wszystkie programy użytkowe

Protokół TCP — własności

- *przesyłanie strumieniami* — dane przesyłane pomiędzy procesami użytkowników traktujemy jako strumień bitów podzielony na oktety — odbiorca otrzymuje dokładnie ten sam ciąg oktetów, które przekazał nadawca
- *łączenie w obwód wirtualny* — pomiędzy nadawcą i odbiorcą ustanawiane jest "połączenie" w celu przesyłania za pomocą strumienia — wymiana informacji jest potwierdzana; błędy (np. zerwanie połączenia) są wykrywane, a programy użtkowe są o nich informowane
- *przesyłanie z użyciem buforów* — w celu zwiększenia efektywności przesyłania i zminimalizowania ruchu w sieci, dane przekazywane przez program użytkowy są buforowane tak, aby wypełniać nimi datagramy o rozsądnej wielkości

Protokół TCP — własności cd.

- *brak strukturalizacji strumienia* — nie ma możliwości dzielenia stumienia na rekordy ani też rozpoznawania zawartości strumienia — programy korzystające z usługi przesyłania strumieniami interpretują zawartość strumienia i zgadzaja się na jego format przed rozpoczęciem transmisji
- *połączenie w pełni dwukierunkowe* (ang. *duplex*) — usługa przesyłania strumieniami TCP/IP umożliwia jednoczesne przesyłanie w obu kierunkach; z punktu widzenia programu użytkowego połączenie składa się z dwóch niezależnych strumieni płynących w przeciwnych kierunkach — może on zatrzymać przepływ w jednym kierunku bez zakłócania przepływu w drugim

Realizacja niezawodności połączenia *pozytywne potwierdzanie z retransmisją:*

- **nadawca** zapamiętuje informacje o każdym wysłanym pakiecie i przed wysłaniem następnego czeka na potwierdzenie
- **nadawca** uruchamia zegar w momencie wysłanie pakietu, a nestępnie wysyła ten pakiet ponownie, jeżeli minie odpowiedni czas, a pakiet nie nadejdzie
- w momencie otrzymania danych **odbiorca** wysyła komunikat potwierdzenia
- w celu rozwiązania problemu z związanego z duplikowaniem, pakiety i potwierdzenia są numerowane

Protokół przesuwających się okien

- Cel:
  - poprawienie efektywności przesyłania strumieniami — prosty protokół z pozytywnym potwierdzaniem powoduje marnowanie przepustowości sieci poprzez oczekiwanie na potwierdzenie każdego pakietu
- Warunki:
  - nadawca numeruje pakiety, aby umożliwić ich potwierdzanie
  - zarówno pakiety jak i ich potwierdzenia mogą być gubione lub duplikowane
  - nadawca i odbiorca utrzymują bufory ("okna"), w których przechowują odpowiednie pakiety

Protokół przesuwających się okien

- Realizacja:
  - **Nadawca**
    - *nadawca* przechowuje kopie wysłanych ramek w *oknie* aż do momentu ich potwierdzenia — jeżeli okno jest pełne, nie odbiera ramek od programu użytkownika
    - odebranie od *odbiorcy* potwierdzenia ramki jest traktowane jako potwierdzenie wszystkich ramek o numerach mniejszych — powoduje "przesunięcie" okna, czyli usunięcie z niego tych ramek
    - brak potwierdzenia odebrania ramki w określonym czasie (*timeout*) powoduje ponowne wysłanie wszystkich ramek o numerach nie mniejszych od tej, znajdujących się w oknie

Protokół przesuwających się okien

- Realizacja — cd. :
  - **Odbiorca**
    * zadaniem *odbiorcy* jest przekazywanie ramek w odpowiedniej kolejności do programu użytkownika
    * *odbiorca* przechowuje niepotwierdzone ramki w swoim "oknie" – odbiera tylko ramki pasujące do "okna", resztę ignoruje
    * odbrane ramki są potwierdzane tylko, gdy wszystkie ramki o numerach mniejszych zostały odebrane – jako potwierdzenie przekazany jest największy numer spośród ramek, które dotarły w dobrej kolejności
    * *odbiorca* przekazuje wszystkie potwierdzone ramki programowi użytkownika

Protokół TCP — co on zapewnia

- specyfikuje format danych i potwierdzeń wymienianych przez komuptery w celu niezawodnej komunikacji
- opisuje procedury używane przez maszyny w celu upewnienia się, że dane dotarły bez zakłóceń
- wyznacza sposób rozróżniania odbiorców na danej maszynie
- opisuje jak maszyny powinny radzić sobie z błędami (utrata pakietu, zduplikowanie)
- określa w jaki sposób dwa komuptery inicjują przesyłanie za pomocą strumienia TCP/IP oraz w jakiej sytuacji zgadzają się na jego zakończenie
- może być używany z różnymi systemami dostarczania pakietów (np. IP, ATM)

Porty, połączenia i końce połączeń

- porty jako mechanizm do identyfikacji w obrebie maszyny
- *punkt końcowy* to para liczb całkowitych *(węzeł, port)*, gdzie węzeł to adres IP, a port to oczywiście port TCP
- podstawowym pojęciem TCP jest *połączenie* identyfikowane przez parę *punktów końcowych*
- wirtualne połączenie pozwala, aby wiele połączeń miało wspólne punkty końcowe — dany numer portu może być przypisany do wielu połączeń na danej maszynie

Pasywne i aktywne otwarcie

- program użytkowy na jednym z końców wykonuje funkcję **pasywnego otwarcia** informując system operacyjny, że akceptuje połączenie, które nastąpi
- program użytkowy na drugim końcu ustanawia połączenie używając funkcji **aktywnego otwarcia**
- po ustanowieniu połączenia programy użytkowe mogą zacząć przekazywać dane

Segmenty, strumienie i numery porządkowe

- TCP dzieli strumień danych (czyli ciąg oktetów) na **segmenty** — każdy segment podróżuje w pojedynczym datagramie IP
- w celu efektywnego przesyłania i kontroli przepływu TCP korzysta z mechanizmu przesuwającego się okna (UWAGA: dzieje się to na poziomie oktetów, a nie segmentów)
- każdy oktet w strumieniu otrzymuje numer porządkowy
- komunikacja w TCP jest dwukierunkowa, zatem każdy z końców połączenie utrzymuje dwa okna (do nadawania i odbierania)

Zmienny rozmiar okna

- TCP umożliwia dynamiczną zmianę rozmiaru okna — każde potwierdzenie zawiera propozycję rozmiaru okna:
  - w odpowiedzi na propozycję wiekszego okna, nadawca zwiększa liczbę oktetów wysyłanych bez potwierdzenia
  - w odpowiedzi na propozycję mniejszego okna, nadawca wstrzymuje wysyłanie oktetów
- używanie okien o zmiennym rozmiarze umożliwia kontrolę przepływu i niezawodne przesyłanie (np. odbiorca może wstrzymać przesyłanie oferując okno rozmiaru 0)

Format segmentów TCP

- nagłówek
  - *port nadawcy* (dwa oktety)
  - *port odbiorcy* (dwa oktety)
  - *numer porządkowy* (cztery oktety) — pozycja danych segmentu w strumieniu
  - *numer potwierdzenia* (cztery oktety) — numer oktetu, który nadawca spodziewa się otrzymać w następnej kolejności
  - *długość nagłówka* (pół oktetu) — długość nagłówka segmentu w wielokrotnościach 32 bitów
  - *zarezerwowane* (trzy czwarte oktetu) — do wykorzystania w przyszłości

Segment TCP cd.

- nagłówek cd.
  - *bity kodu* (trzy czwarte oktetu) bity od lewej do prawej
    * URG — wskaźnik pilości jest istotny
    * ACK — pole potwierdzenia jest istotne
    * PSH — segment stanowi prośbę o wypchnięcie
    * RST — skasuj połączenie
    * SYN — zsynchronizuj numery porządkowe
    * FIN — koniec strumienia bajtów u nadawcy
  - *okno* (dwa oktety) — proponowany rozmiar okna
  - *suma kontrolna* (dwa oktety) — obliczana na podstawie segmentu z dodanym psudonagłówkiem (adres IP nadawcy, adres IP odbiorcy, zero, protokół i długość TCP)

Segment TCP cd.

- nagłówek cd.
  - *wskaźnik pilnych danych* (dwa oktety) — gdy ustawiony jest wskaźnik URG, jest to miejsce danych pilnych w segmencie (dane te należy odebrać poza głównym strumieniem transmisji)
  - *opcje (jeżeli jakieś)* (do trzech oktetów) — używane do negocjowania pomiędzy nadawcą i odbiorcą maksymalnego rozmiaru segmentu TCP (najlepiej, aby datagramy IP przenoszące segmentu były możliwie duże, ale nie następowała ich fragmentacja)
  - *uzupełnienie* (co najmniej jeden oktet) — uzupełnienie opcji do wielkości czterech oktetów
- pole danych o zmiennej długości

Skumulowane potwierdzanie

- potwierdzenie określa wartość numeru porządkowego o jeden wiekszą niż wartość najdalszego oktetu w ciągłym prefiksie odebranych oktetów
- zalety
  - potwierdzenia są łatwe do wykonania i jednoznaczne
  - utrata potwierdzenia nie musi powodować retransmisji
- wady
  - brak informacji o wszytkich poprawnych transmisjach może powodować nieefektywność retransmisji

Retransmisja z adaptacją

- transmisja pakietu wiąże się z ustawieniem dla niego czasu oczekiwania na potwierdzenie — czas osiągania odbiorcy może zmieniać się w czasie, w zależności od wahań natężenia ruch lub zmiany trasy transmisji
- próba pierwsza:
  $RTT = \alpha * Stare\_RTT + (1 - \alpha) * Probka$,
  gdzie
  - RTT — czas podróży w obie strony
  - $Probka$ — ostatnio uzyskana próbka czasu
  - $0 \leq \alpha \leq 1$ — waga

  $Czas\_oczekiwania = \beta * RTT$, gdzie
  - β — waga, pierwotnie wartość 2

Algorytm Karna

- **niejednoznaczność potwierdzenia** — w przypadku retransmisji nie można sprawdzić, czy potwierdzenie dotyczy pierwotnego, czy retransmitowanego datagramu — może prowadzić do błędnych oszacowań czasu oczekiwania
  $Nowy\_czas\_ocz = \gamma * Czas\_oczekiwania$, gdzie
  - γ — mnożnik, zwykle wartość 2
- Karna: *przy obliczaniu czasów oczekiwania ignoruj próbki odpowiadające retransmitowanym segmentom, ale używaj strategii wydłużania czasów oczekiwania i nie uwzględniaj wartości czasu oczekiwania uzyskanych z retransmitowanych pakietów, dopóki nie uzyskasz poprawnej odpowiedzi*

Reakcja na dużą wariancję

- wartość β równa 2 daje dobre dopasowanie czasów oczekiwania dla obciążeń ok 30% — trzeba czegoś lepszego:
  $Rozn = Probka - Stare\_RTT$
  $Popr\_RTT = Stare\_RTT + \delta * Rozn$
  $Odch =$
  $Stare\_Odch + \rho(|Rozn| - Stara\_Rozn)$
  $Czas\_oczekiwania = Popr\_RTT + \eta * Odch$, gdzie
  - $Odch$ — szacowana średnia wartość odchylenia
  - $0 \leq \delta \leq 1$ — jak nowe próbki wpływają na średnią
  - $0 \leq \rho \leq 1$ — jak nowe próbki wpływają na średnie odchylenie
  - η — jak odchylenie wpływa na czas oczekiwania
- zazwyczaj $\delta = 1/8$, a $\rho = 1/4$

Unikanie przeciążenia

- *unikanie przeciążenia przez wielokrotne zmniejszanie* — gdy następuje zgubienie segmentu zredukuj okno przeciążeniowe o połowę (jeżeli jeszcze nie jest równe 1); dla tych segmentów, które pozostały w dozwolonym oknie, zwiększ wykładniczo czas transmisji
  $Dozw\_okno = min(prop\_odb, okno\_przec)$
- *powolny start* — przy rozpoczynaniu ruchu w ramach nowego połączenia albo przy zwiększaniu ruchu po przeciążeniu, ustaw okno przeciążeniowe w wielkość jednego segmentu i zwiększaj je za każdym razem, gdy przychodzi potwierdzenie
- aby uniknąć dodatkowego obciążenia z chwilą, gdy okno przeciążeniowe osiąga połowę swojego pierwotnego rozmiaru TCP zmniejsza szybkość zwiększania okna — okno zwiększane jest tylko o 1

Otwieranie połączenia TCP

- maszyna losowo wybiera numer porządkowy w strumieniu, używany do identyfikowania bajtów w strumieniu, który je niesie
- nadawca (inicjator) wysyła segment **SYN** z ustawionym bitem SYN w polu kodu oraz z wylosowanym uprzednio numerem porządkowym x
- odbiorca odbiera segment **SYN** i wysyła swój segment **SYN**, z ustawionym bitem ACK dla $x + 1$ i ze swoim numerem porządkowym y
- nadawca odbiera segment **SYN+ACK** i wysyła ACK dla $y + 1$
- odbiorca odbiera segment **ACK**

Zamykanie połączenia TCP

- – program użytkowy na węźle 1 zamyka połączenie — wysyłany jest segment **FIN** z ustawionym bitem FIN i numerem x
  – węzeł 2 odbiera segment **FIN**, informuje swój program użytkowy i wysyłą segment **ACK** dla $x + 1$
  – pierwszy odbiera segment **ACK** i połączenie w jedną stronę jest zamknięte
  – na węźle 2 program użytkowy zamyka połączenie, węzeł wysyła **FIN+ACK** z numerem y i bitem ACK dla $x + 1$
  – węzeł 1 odbiera segment **FIN+ACK** i wysyła **ACK** dla $y + 1$
  – drugi odbiera segment **ACK**
- w sytuacjach wyjątkowych można skorzystać z segmentu z bitem RST do natychmiastowego przerwania (skasowania) połączenia

Syndrom “głupiego” okna

- pojawia się gdy każde potwierdzenie proponuje małą ilość miejsca (np. odbiorca mając pełne okno odbiera dane po jednym oktecie i potwierdza je proponując okno wielkości 1), w związku z czym każdy segment przenosi małą ilość danych — marnuje się przepustowość sieci i wprowadza dodatkowe obliczenia
- unikanie po stronie odbiorcy: gdy zaoferowano zerowe okno, to przed wysłaniem aktualnej oferty okna należy poczekać, aż stanie się dotępne miejsce równe wielkości conajmniej 50% całkowitej pojemności bufora lub równe maksymalnemu rozmiarowi segmentu — opóźnione potwierdzanie

Syndrom “głupiego” okna cd.

- unikanie po stronie nadawcy — metoda **samotaktująca**: gdy program wysyłający generyje dodatkowe dane do przesyłania przez połączenie, którym uprzednio już były wysyłane dane, ale nie przyszło dla nich potwierdzenie, umieść te nowe dane jak zwykle w buforze wyjściowym, ale nie wysyłaj dodatkowych segmentów, zanim danych nie będzie na tyle dużo, aby stanowiły segment maksymalnej wielkości; jeżeli w momencie przyjścia potwierdzenia wciąż jeszcze czekasz na wysłanie, to wyślij zgromadzone dane; stosuj tę zasadę nawet wtedy, gdy użytkownik zażąda operacji wypchnięcia — *algrytm Nagle’a*
- obecnie TCP wymaga aby nadawca i odbiorca implementowali powyższe heurystyczne algorytmy unikania syndromu “głupiego” okna

Pytania do sali

- Dlaczego zgubienie potwierdzenia nie musi powodować retransmisji ?
- Jakie są argumenty za i przeciw automatycznemu zamykaniu nie używanych połączeń ?
- Do czego można użyć mechnizmu wypychania i danych pilnych ?
- Czy protokół TCP nakłada górne ograniczenie na przepływność danych ?
- Jakie są wady i zalety opóźnionego potwierdzania ?