INSTYTUT BADAŃ SYSTEMOWYCH
POLSKIEJ AKADEMII NAUK

ANALIZA SYSTEMOWA W FINANSACH I ZARZĄDZANIU

Wybrane problemy
Tom 11

Pod redakcją
Jerzego HOŁUBCA

Warszawa 2009

INSTYTUT BADAŃ SYSTEMOWYCH
POLSKIEJ AKADEMII NAUK

ANALIZA SYSTEMOWA W FINANSACH I ZARZĄDZANIU

Wybrane problemy
Tom 11

Pod redakcją
Jerzego HOŁUBCA

Warszawa 2009

Wykaz opiniodawców artykułów zamieszczonych w niniejszym tomie:

prof. dr hab. inż. Jerzy HOŁUBIEC
dr inż. Lech KRUŚ
doc. dr hab. inż. Wiesław KRAJEWSKI
doc. dr hab. Jacek MALINOWSKI
dr inż. Edward MICHALEWSKI
prof. dr Adam SKOREK
dr hab. Ryszard SMARZEWSKI
prof. dr hab. inż. Andrzej STRASZAK
dr Dominik ŚLĘZAK
prof. dr hab. inż. Stanisław WALUKIEWICZ
doc. dr hab. Sławomir ZADROŻNY

Warszawa 2009

ISBN 9788389475220

Druk: Zakład Poligraficzny Jerzy Kosiński, Warszawa

ADAPTACJA W ZARZĄDZANIU NATŁOKIEM PROTOKOŁU TCP

Sebastian Konkol
Studia Doktoranckie IBS PAN

Artykuł prezentuje protokół TCP przez pryzmat adaptacji – dostosowania parametrów transmisji do zmiany warunków połączenia. Wskazane są zarówno zagadnienia zachowań sieci jak i węzłów końcowych połączenia. Podsumowano możliwości adaptacji realizowane przez algorytmy kontroli natłoku i wskazano kierunki badawcze.

***Słowa kluczowe**: TCP, zarządzanie natlokiem, adaptacja, DropTail, RED, ECN, REM, Tahoe, Reno, Vegas, Westwood*

This article presents TCP protocol in terms of its adaptability – adjusting transmission parameters based on connection state. Areas of network behaviour and end-nodes have been depicted together with their role in TCP congestion control. Adaptation capabilities carried by congestion control algorithms have been summarized and further research areas have been portrayed.

***Keywords**: TCP, congestion avoidance and control, adaptability, DropTail, RED, ECN, REM, Tahoe, Reno, Vegas, Westwood*

Wprowadzenie

TCP jest najczęściej wykorzystywanym protokołem komunikacyjnym w sieci Internet. Według różnych pomiarów (Miller, Thompson, 1998; Leinen, Schulzrinne, 2001) przesłanie 95% bajtów i 90% pakietów oraz zestawienie 80% połączeń w sieci Internet jest możliwe dzięki mechanizmom protokołu TCP. Na bazie tych mechanizmów funkcjonować mogą aplikacje WWW, FTP, poczty elektronicznej czy list dyskusyjnych głęboko już zakorzenione w codziennym życiu człowieka i społeczności – zarówno prywatnym jak i zawodowym. Konsekwencją tej zależności są wysokie i cały czas podnoszone wymagania stawiane przed jakością i niezawodnością komunikacji, wśród których dużą rolę odgrywają algorytmy zarządzania natłokiem.

Niniejsze opracowanie przybliża stosowanie koncepcji adaptacji w mechanizmach zarządzania natłokiem protokołu TCP. W rozdziale 1. prezentowane są kluczowe cechy protokołu TCP pozwalające na adaptację charakterystyki transmisji.

Rozdział 2. przedstawia model komunikacji w sieci między nadawcą a odbiorcą – środowisko realizacji połączenia TCP. W rozdziale 3. prezentowane są adaptacyjne cechy zachowania węzłów sieci TCP, a w rozdziale 4. omawiane są mechanizmy adaptacji różnych wersji protokołu TCP. Rozdział 5. przedstawia podsumowanie w postaci wniosków i planów badawczych nad rozszerzeniem możliwości adaptacji w zarządzaniu natłokiem protokołu TCP.

1. Charakterystyka transmisji TCP

TCP jest protokołem komunikacyjnym trzeciej warstwy stosu protokołów internetowych. Jego podstawową usługą jest realizacja transmisji strumienia bitów między dwoma węzłami sieci IP. Dla świadczenia takiej usługi TCP bazuje na usługach protokołu IP, który umożliwia przesyłanie pakietów danych między dwoma węzłami sieci bez gwarancji ich dostarczenia, ani zapewnienia poprawnej sekwencji przesłania serii pakietów przez sieć. Mechanizmy TCP zostały więc skonstruowane tak, aby podnieść niezawodność komunikacji (retransmisja utraconych pakietów) i zapewnić strumieniowy charakter przesyłanych danych (odtwarzanie sekwencji pakietów u odbiorcy). Dzięki takiej konstrukcji, protokół TCP umożliwia realizację transmisji danych aplikacyjnych w trybie pseudo połączeniowym bazując na bezpołączeniowym protokole IP.

Poza podstawową transmisją danych, system komunikacyjny tworzący protokół TCP realizuje inne funkcje w kilku obszarach, m. in.: sterowania przepływem, multipleksowania strumieni komunikacji, obsługi zestawiania połączeń i zarządzania nimi oraz pierwszeństwa i bezpieczeństwa.

TCP zapewnia transfer ciągłego strumienia danych pomiędzy parą komunikujących się procesów (w obu kierunkach niezależnie) poprzez umieszczanie pewnej porcji danych w segmentach przeznaczonych do transmisji przez sieć Internet. Odebranie każdej porcji danych wysyłanych przez nadawcę do odbiorcy musi zostać potwierdzona przez odbiorcę poprzez odesłanie zwrotne pakietu potwierdzenia identyfikującego paczkę danych, których otrzymanie jest potwierdzane. TCP zapewnia odtworzenie stanu transmisji w przypadku różnych zdarzeń w systemie komunikacyjnym sieci Internet: uszkodzenia, utraty, zduplikowania pakietów lub ich dostarczenia w zaburzonej kolejności. Jest to osiągane dzięki logicznemu przyporządkowaniu numeru sekwencyjnego każdemu transmitowanemu oktetowi oraz wymaganiu pozytywnego potwierdzenia otrzymania od odbiorcy TCP. W przypadku nieotrzymania przez nadawcę potwierdzenia w z góry określonym czasie (ang. *timeout*) niepotwierdzone dane są retransmitowane przez nadawcę. Po stronie odbiorcy numery sekwencyjne służą odtworzeniu poprawnej kolejności i eliminacji zduplikowanych segmentów.

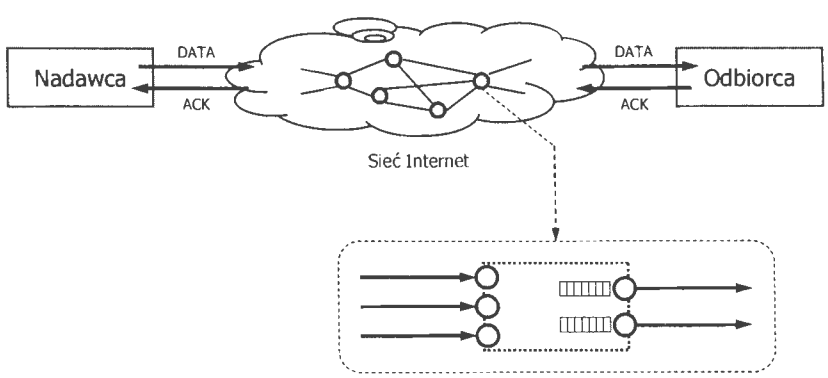
TCP wprowadza także mechanizm sterowania przepływem (ang. *flow control*), dzięki któremu odbiorca może wpływać na szybkość transmisji realizowanej przez nadawcę. Jest to osiąganc przez zwrotne przekazywanie przez odbiorcę do nadawcy w każdym pakiecie potwicrdzenia rozmiaru okna danych określającego akceptowalny zakres numerów sekwencyjnych poza ostatni segment poprawnie odebrany (ang. *advised window*). Rozmiar tego okna określa maksymalną liczbę oktetów, jakie nadawca może wysłać do odbiorcy przed otrzymaniem dalszego „pozwolenia" ze strony odbiorcy (potwierdzenia poprawnego odebrania części przesłanych danych). Sterowanie przepływem danych w protokole TCP realizowane jest po stronie nadawcy i opiera się o koncepcję ślizgającego okna (ang. *sliding window*) – nadawca wysyła do odbiorcy dane o wielkości nie przekraczającej rozmiaru okna dozwolonego przez odbiorcę, po czym przechodzi w stan oczekiwania na potwierdzenie odebrania ze strony odbiorcy. Po uzyskaniu tego potwierdzenia nadawca może wysłać do odbiorcy nową porcję danych o takiej wielkości, aby wolumen danych wysłanych i nicpotwierdzonych nie przekroczył wielkości dozwolonego okna odbiorcy. Jako potwicrdzenia poprawnego odebrania danych – określenia po stronie nadawcy, które z wysłanych pakietów danych (pakiet typu DATA) dotarły do odbiorcy – wykorzystywanc są pakiety potwierdzeń (pakiet typu ACK), w jakich odbiorca umieszcza numer sekwencyjny kolejnego pakietu danych, jakiego odebrania się spodziewa.

Mcchanizmy zarządzania natłokiem protokołu TCP zostały powiązane ze sterowaniem przepływem danych w taki sposób, że rozmiar okna transmisji (*sliping window*) jest uzależniane nie tylko od rozmiaru okna odbiorcy (*advised window*), ale także rozmiaru okna natłoku (ang. *congestion window*). Ta ostatnia wartość jest systematycznie wyliczana przez algorytm zarządzania natłokiem po stronie nadawcy zaimplementowany w danej wersji protokołu TCP, a dozwolony w danej chwili rozmiar okna transmisji jest wyliczany jako minimum rozmiaru okna odbiorcy i okna natłoku. Wprowadzenie takiej zależność stwarza możliwości podstawowej adaptacji zachowania protokołu TCP (szybkość transmisji) do warunków transmisji panujących w sieci.

2. Komunikacja w sieci

Komunikacja oparta o protokół TCP zachodzi między dwoma aplikacjami funkcjonującymi w sieci Internet, zwykle w oddzielnych systemach (Rys. 1. Schemat komunikacji TCP). Komunikacja TCP jest dwukierunkowa (ang. *full duplex*) co oznacza, że strumień danych użytkowych (pakiety typu DATA) może być niezalcżnie wysyłany zarówno przcz jeden system i odbierany przez drugi, jak i w kierunku przeciwnym. Dla uproszczcnia rozważań, ograniczymy się tu do jednego kierunku ruchu użytkowego, a w takiej relacji system wysyłający będzie na-

zywany nadawcą, a system docelowy – odbiorcą. Dla realizacji skutecznej transmisji danych w protokole TCP konieczne jest poprawne przejście przez sieć pakietu danych (pakiet typu DATA) od nadawcy do odbiorcy oraz poprawne przejście pakietu potwierdzenia od odbiorcy do nadawcy (pakiet typu ACK). Przesyłanie pakietów danych między nadawcą a odbiorcą oraz pakietów potwierdzeń między odbiorcą a nadawcą realizowane jest przy wykorzystaniu infrastruktury sieci. Wnętrze sieci składa się z węzłów pośrednich (węzłów sieci) połączonych łączami transmisyjnymi. Z perspektywy protokołu TCP każdy węzeł sieci może być przedstawiony jako system obsługi z kolejkami o skończonych pojemnościach, przeznaczonych do buforowania danych przesłanych przez każde z łączy wychodzących z tego węzła. Każde łącze w sieci może być z tej samej perspektywy przedstawione jako kanał transmisyjny o skończonej przepustowości dzielonej między różne połączenia oraz o niezerowym czasie transmisji danych (czasie propagacji) i innych parametrach wpływających na transmisję jak np. stopę błędów transmisji. Miarą sumarycznego czasu skutecznej komunikacji – przesłania pakietu DATA i odebrania pakietu ACK – jest RTT (ang. *Rount-Trip Time*), a wartość tej miary zmienia się w czasie wraz ze zmianami stanu sieci (głównie wynikającym z czasu oczekiwania na transmisję w buforach węzłów sieci) oraz wyborem drogi w sieci, jaką przesyłane są poszczególne pakiety.



Rys. 1. Schemat komunikacji TCP

W chwili inicjowania połączenia TCP nadawca nie posiada wiedzy na temat dwóch istotnych ograniczeń dla szybkości transmisji: wielkości danych, na przyjęcie których przygotowany jest odbiorca oraz stanu sieci wyznaczonego przez obciążenie poszczególnych jej węzłów. Wartość pierwszego ograniczenia jest udostępniana nadawcy w ramach działania mechanizmu sterowania przepływem (*advised window*) jest więc dostępna dla nadawcy explicite, a w skali czasu jednego połączenia pozostaje raczej niezmienna. Drugie ograniczenie – stan sieci, na który

składają się warunki pracy wszystkich węzłów i łączy z jakich korzysta połączenie transmitując dane – jest informacją niedostępną explicite dla nadawcy i silnie zmienną w skali czasu trwania połączenia TCP. Dla osłabienia wpływu braku tej wiedzy nadawca TCP rozpoczyna transmisję danych z małą prędkością i stopniowo zwiększa ją (zwiększanie rozmiaru okna natłoku) aż do momentu detekcji sygnałów stanu natłoku w sieci. W reakcji na wystąpienie takiego sygnału następuje znaczące zmniejszenie szybkości transmisji (redukcja rozmiaru okna natłoku) pozwalające z dużym prawdopodobieństwem na wyjście ze stanu natłoku, po czym szybkość transmisji jest ponownie stopniowo zwiększana do ponownego wystąpienia kolejnego sygnału natłoku. Zachowanie mechanizmu zarządzania natłokiem TCP po stronie nadawcy polegające na „dostrajaniu" chwilowej szybkości transmisji do obserwowanych cech opisujących stan sieci zapewnia podstawową adaptację parametrów połączenia TCP do stanu sieci. Adaptacja ta ma także na celu utrzymanie możliwie wysokiego wykorzystania dostępnego pasma przez połączenie TCP w obliczy zmian stanu sieci.

Wiele informacji, jakie docierają do nadawcy lub mogą być przez nadawcę dedukowane na podstawie innych obserwacji, może pełnić rolę sygnałów informujących o stanie natłoku. Tradycyjnie, za sygnalizację natłoku przyjmuje się otrzymanie przez nadawcę powtórzonych potwierdzeń (czyli kolejnych pakietów potwierdzeń tego samego pakietu danych) oraz przekroczenie dopuszczalnej wartości czasu oczekiwania na odebranie potwierdzenia (*timeout*). W propozycjach tworzonych w trakcie rozwoju mechanizmów zarządzania natłokiem w sieci TCP za taką sygnalizację przyjmowało się rozważań także wydłużenie czasu RTT, czy stosowanie dodatkowej sygnalizacji explicite wskazującej stan natłoku (ang. *ECN – Explicit Congestion Notification*). Większość z tych sygnałów niesie informację binarną (stan natłoku jest lub go nie ma), ale nieliczne dostarczać mogą bogatszej informacji – nasilenia zjawiska natłoku lub prawdopodobieństwa jego wystąpienia w najbliższym czasie. Choć podstawowe mechanizmy zarządzania natłokiem implementowane są po stronie nadawcy TCP, istnieje pewna klasa sygnałów, które mogą być interpretowane przez węzły pośrednie (węzły sieci) jako sygnały stanu natłoku w danym węźle, a odpowiednie reakcje na takie sygnały mogą wspomagać działania zarządzania natłokiem.

3. Adaptacja w zachowaniu sieci

Węzły pośrednie są złożone z systemów wejściowych obsługujących dane przychodzące do tego węzła oraz systemów wyjściowych obsługujących relacje wychodzące. Sygnały powstawania stanu natłoku w węźle pośrednim mogą być obserwowane w systemach obsługujących wyjściowe kierunki węzła sieci. Każdy kierunek wyjściowy węzła sieci to system z jednym stanowiskiem obsługi i kolejką

o skończonej długości. Podstawowym sygnałem zaistnienia stanu natłoku w węźle pośrednim jest przepełnienie bufora danych (kolejki systemu) dla łącza, którym pakiet miałby być przesłany. W takiej sytuacji węzeł sieci po prostu odrzuca pakiet, który „nie zmieścił się" w buforze, nie informując o tym żadnego innego węzła. Takie zachowanie, nazywane DropTail, nie powstało w wyniku świadomych decyzji projektowych, ale jako „efekt uboczny" sposobu funkcjonowania węzłów sieci. W wyniku przeprowadzenia szeregu badań analitycznych i symulacyjnych takiego schematu obsługi udowodniono jego szkodliwość nie tylko dla połączenia, którego dotyka odrzucenie pakietu, ale dla całości transmisji realizowanej przez ten węzeł. Szkodliwość ta objawia się efektem synchronizacji faz strat pakietów dla dużej części połączeń TCP wychodzących przez to samo łącze – skoro bufor jest zapełniony, to żaden pakiet (niezależnie od pochodzenia) nie zostanie przyjęty, a wszystkie połączenia TCP wychodzące przez to łącze utracą przesłane pakiety. Brak miejsca w buforze danych, powinien być więc ostateczną informacją dla węzła na zaistnieniu stanu natłoku, nie zaś zaledwie pierwszą dostępną. Takie wnioski zainicjowały badania nad metodami aktywnego zarządzania kolejkami systemów obsługi w węzłach sieci (ang. *AQM – Active Queue Management*) – metodami, które działałyby poprawnie w ograniczeniach konstrukcyjnych sieci Internet (autonomia węzłów), ale lepiej współpracowałyby w zakresie zarządzania natłokiem w sieci i byłyby pozbawione wad schematu DropTail.

Zgodnie z najpowszechniej obecnie stosowaną metodą AQM, RED (ang. *Random Early Detection*), węzeł sieci przechowuje dwie wartości progowe zajętości kolejki (dolną i górną) oraz uzależnia przyjęcie pakietu do kolejki od poziomu zajętości kolejki w danej chwili. Jeśli zajętość kolejki jest mniejsza niż dolna wartość progowa, węzeł bezwarunkowo przyjmuje pakiet do kolejki. Jeśli zajętość plasuje się między dolną i górną wartością progową, węzeł odrzuca pakiet z prawdopodobieństwem tym większym im większa jest zajętość kolejki. Powyżej górnej wartości granicznej każdy pakiet jest bezwarunkowo odrzucany. Co prawda konsekwencją takiego działania jest nadal odrzucenie pakietu bez informowania o tym węzłów końcowych, ale odrzucany jest zwykle pakiet jednego połączenia TCP – stan natłoku jest niejako symulowany dla tego połączenia TCP, a pozostałe połączenia korzystające z tego samego łącza pracują dalej. Dzięki wprowadzeniu czynnika losowości eliminowany jest efekt synchronizacji fazowej połączeń i zmniejszane jest prawdopodobieństwo stanu przeciążenia.

Ewolucja sposobu myślenia o roli węzłów pośrednich w realizacji zarządzania natłokiem doprowadziła także do wprowadzenia możliwości jawnej sygnalizacji stanu natłoku w ścieżce wykorzystywanej przez dane połączenie TCP – powstania mechanizmu ECN (ang. *Explicit Congestion Notification*). ECN wykorzystuje część nagłówka pakietu IP dla zapisania informacji o stanie natłoku w postaci informacji o szerszej dziedzinie wartości (rozróżnienie sytuacji braku natłoku, zbli-

żania się i istnienia takiego stanu). Takie oznaczenie propagowane jest przez sieć do węzłów końcowych. Taki mechanizm został zaproponowany w algorytmie REM (ang. *Random Exponential Marking*). W myśl tego algorytmu wyznaczana jest decyzja o przyjęciu do kolejki analogicznie jak w algorytmie RED, ale zamiast rzeczywistego odrzucenia w węźle sieci decyzja ta jest propagowana do nadawcy zgodnie z zasadami rządzącymi mechanizmem ECN. Takie zachowanie umożliwia nadawcy reakcję na narastający stan natłoku (zmniejszenie szybkości transmisji) bez utraty pakietu i konieczności retransmisji.

Zachowanie poszczególnych algorytmów wykazuje cechy adaptacji do zmieniających się warunków w sieci, ale także wspomaga funkcje adaptacji realizowane przez podstawowe algorytmy zarządzania natłokiem w węzłach końcowych. Opis matematyczny zachowania mechanizmów wspomagających zarządzanie natłokiem w sieci bazuje na modelowaniu stanu ustalonego i metodzie opisu „kosztu" jako miary natłoku (Kelly, Maulloo, Tan, 1998). Tabela poniżej prezentuje wzory wyliczania „kosztów" dla poszczególnych algorytmów (Low, Paganini, Doyle, 2002; Athuraliya, Li, Law, Yin, 2001), gdzie $y_i(t)$ oznacza chwilowy sumaryczny wolumen danych przesyłanych przez i-te łącze, c_i – wielkość bufora danych i-tego łącza, natomiast γ jest parametrem algorytmu REM.

Algorytm	**Koszt**
DropTail	Czas kolejkowania: $\dot{p}_i(t) = \begin{cases} \frac{1}{c_i}(y_i(t) - c_i) & p_i(t) > 0 \\ \frac{1}{c_i}[y_i(t) - c_i]^+ & p_i(t) = 0 \end{cases}$
RED	Długość kolejki: $\dot{b}_i(t) = \begin{cases} (y_i(t) - c_i) & b_i(t) > 0 \\ [y_i(t) - c_i]^+ & b_i(t) = 0 \end{cases}$
REM	Długość kolejki: $b_i(t+1) = [b_i(t) + \gamma(y_i(t) - c_i)]^+$

4. Podstawowa adaptacja w zachowaniu nadawcy

Pierwotnym pomysłem wykrywania stanu natłoku po stronie nadawcy było dokonywanie przez źródło próbkowania dostępnej, wolnej przepływności w sieci poprzez liniowe zwiększanie maksymalnego dopuszczalnego rozmiaru okna transmisji aż do momentu wykrycia sygnału natłoku i eksponencjalnym zmniejszeniu rozmiaru tego okna w odpowiedzi na taki sygnał. Stan natłoku był pierwotnie utożsamiany z wykryciem utraty pakietu, przy czym objawami takiej sytuacji mogły być zarówno odebranie zduplikowanych pakietów potwierdzeń jak i przekroczenie granicznego czasu oczekiwania na potwierdzenie (*timeout*).

Taka idea została zaimplementowana jako pierwsza wersja protokołu TCP posiadająca mechanizm zarządzania natłokiem – TCP Tahoe (Jacobson 1988). Algorytm ten działa w jednym z dwóch stanów – *slow-start* albo *congestion avoidance*. Każde zestawiane połączenie jest rozpoczynane w stanie *slow-start*, w którym rozmiar okna natłoku (*congestion window*) ustawiany jest na wielkość jednego pakietu, a nadawca może go zwiększyć o jeden po każdym odebraniu pakietu potwierdzenia – pozwalało to na podwojenie rozmiaru okna po każdym czasie RTT. Takie zwiększanie rozmiaru okna ma miejsce do momentu przekroczenia progu ustalonego dla stanu *slow-start* – po przekroczeniu tego progu algorytm zarządzania natłokiem przechodzi do stanu *congestion avoidance*. Wartość progu jest ustalana arbitralnie w czasie zestawiania połączenia, ale ma wpływ na zachowanie połączenia jedynie w czasie pierwszego przebywania w stanie *slow-start*, gdyż jest aktualizowana po każdym wystąpieniu natłoku. W stanie *congestion avoidance* rozmiar okna jest zwiększany o wartość równą odwrotności aktualnego rozmiaru okna po odebraniu każdego poprawnego pakietu potwierdzenia, co skutkuje zwiększeniem rozmiaru okna o jeden pakiet w odstępie czasu równym RTT. Zarówno w stanie *slow-start* jak i *congestion avoidance* wykrycie utraty pakietu wyzwalało ustalenie wielkości progu dla stanu *slow-start* na połowę aktualnego rozmiaru okna, retransmisję utraconego pakietu i przejście do stanu *slow-start* z ustawieniem rozmiaru okna na wartość jednego pakietu.

Poniższy pseudokod ilustruje sposób działania TCP Tahoe. W tym algorytmie `cwnd` oznacza rozmiar okna natłoku, `ssthresh` oznacza poziom graniczny dla stanu *slow-start* (wielkości wyrażone w oktetach). Wielkość `seg_size` oznacza rozmiar segmentu danych i jest wartością stałą wyrażoną w oktetach. Tahoe reaguje na dwie klasy zdarzeń: odebranie poprawnego potwierdzenia dostarczenia pakietu danych (ACK) oraz utratę pakietu (3 powtórzone ACK lub upłynięcie granicznego czasu oczekiwania na potwierdzenie).

```
INIT:
   cwnd := seg_size
   ssthresh ← default value > seg_size
ACK:
   if (cwnd <= ssthresh) /* slow-start phase */
      cwnd := cwnd + seg_size
   else /* congestion avoidance phase */
      cwnd := cwnd + 1/cwnd
   endif
DUPACK or TIMEOUT: /* packet loss */
   ssthresh := cwnd/2
   cwnd := seg_size
```

W nowszym algorytmie wersji, TCP Reno (Alman, Paxson, Stevens, 1999), które jest obecnie najpopularniejszym algorytmem zarządzania natłokiem w sieci Internet, rozszerzono mechanizmy zarządzania natłokiem, o identyfikację i obsługę fazy szybkiego odtworzenia (ang. *fast recovery*). Faza ta jest określona jako czas od momentu detekcji utraty pakietu (dwukrotne otrzymanie potwierdzenia tego samego pakietu) do czasu odebrania potwierdzenia odebrania pakietu, który pierwotnie został utracony. Na obsługę tej fazy działania mechanizmu zarządzania natłokiem składają się dwie modyfikacje pozwalające na zwiększenie efektywności odtwarzania stanu połączenia po utracie pakietu. W algorytmie TCP Tahoe rozmiar okna jest zamrażany na okres odtworzenia stanu połączenia po utracie pakietu, więc kolejny pakiet może być przesłany dopiero po upłynięciu czasu RTT, a podczas realizacji transmisji tego pakietu sieć jest „oczyszczana" z pakietów tego połączenia. W wyniku takiego zachowania część węzłów pośrednich pozostaje bezczynnymi, co obniża efektywność wykorzystania zasobów sieci. Pierwsza modyfikacja TCP Reno w fazie *fast recovery* pozwalała na tymczasowe zwiększenie rozmiaru okna o jeden pakiet po każdym otrzymanym podwójnym potwierdzeniu – oznacza to bowiem, że pakiet został poprawnie przetransmitowany przez sieć. W sytuacji, gdy rozmiar okna w ten sposób powiększony stawał się większy niż liczba pakietów w sieci, kolejny pakiet mógł być przesłany w stanie *fast recovery* w oczekiwaniu na poprawne, pojedyncze potwierdzenie retransmitowanego pakietu. Druga modyfikacja wprowadzała ustawienie w chwili wyjścia ze stanu *fast recovery* rozmiaru okna na połowę rozmiaru okna z początku tej fazy oraz nakazywała przejście bezpośrednio do stanu *congestion avoidance* z pominięciem stanu *slow-start*. Dzięki temu połączenie kontrolowane przez algorytm TCP Reno wchodziło w stan *slow-start* jedynie zaraz po zestawieniu połączenia albo upłynięciu granicznego czasu oczekiwania na potwierdzenie (*timeout*), nie zaś w przypadku wielokrotnego odebrania potwierdzenia tego samego pakietu danych.

Poniższy pseudokod ilustruje sposób działania TCP Reno. W tym algorytmie `cwnd` oznacza rozmiar okna natłoku, `ssthresh` oznacza poziom graniczny dla stanu *slow-start* (wielkości wyrażone w oktetach), a `fast_recovery` jest zmienną lo-

giczną wyznaczającą fazę *fast recovery*. Wielkość `seg_size` oznacza rozmiar segmentu danych i jest wartością stałą wyrażoną w oktetach. Reno reaguje na trzy klasy zdarzeń: odebranie poprawnego potwierdzenia dostarczenia pakietu danych (ACK), odebranie powtórzonego potwierdzenia dostarczenia pakietu danych oraz utratę pakietu (3 powtórzone ACK lub upłynięcie granicznego czasu oczekiwania na potwierdzenie). Reno zachowuje więc trzon funkcjonalności Tahoe dodając obsługę fazy *fast recovery* podnoszącą sprawność algorytmu.

```
INIT:
   cwnd := seg_size
   ssthresh ← default value > seg_size
   fast_recovery := false
ACK:
   if (cwnd <= ssthresh) /* slow-start phase */
      cwnd := cwnd + seg_size
   else /* congestion avoidance phase */
      cwnd := cwnd + 1/cwnd
   endif
2 DUPACK:
   if (fast_recovery = false) /* fast recovery phase detected */
      fast_recovery := true
      fr_wnd := cwnd/2
   else /* fast recovery phase */
      if (last retransmitted packed ACKed)
         fast_recovery := false
         cwnd := fr_wnd
      else
         cwnd := cwnd + seg_size
      endif
   endif
3 DUPACK or TIMEOUT: /* packet loss */
   ssthresh := cwnd/2
   cwnd := seg_size
```

W algorytmie implementowanym w wersji TCP o nazwie Vegas (Brakmo, Peterson, 1995) w celu detekcji narastania stanu natłoku wykorzystywane są dane o większej dziedzinie wartości. TCP Vegas opiera się na zasadach funkcjonowania TCP Reno i wprowadza usprawnienia przy wykorzystaniu trzech technik. Pierwsza z nich odnosi się do mechanizmu retransmisji, w którym sprawdzanie upłynięcia granicznego czasu oczekiwania wykonywane jest w momencie otrzymania zduplikowanego potwierdzenia, zamiast oczekiwania na trzecie powtórzenie tego potwierdzenia co skutkuje szybszym wykrywaniem utraty pakietu. Druga technika wprowadza rozważniejszy sposób powiększania okna transmisji w stanie *slow-start* uzyskując mniej strat w tym stanie – TCP Tahoe i TCP Reno zwiększają rozmiar okna jest wykładniczo, co przy niewłaściwie oszacowanym poziomie granicznym dla fazy *slow-start* może prowadzić do dużych strat. Trzecia technika to nowy mechanizm zapobiegania natłokowi, który poprawia oscylacyjne zachowanie TCP

Reno wynikające z ustalenia punktu pracy na granicy natłoku w sieci. Pomysł polega tu na dokonywaniu przez źródło estymacji liczby własnych pakietów „zbuforowanych" przez sieć i próbie takiego modyfikowania maksymalnego dozwolonego rozmiaru okna transmisji, aby liczba zbuforowanych pakietów pozostawała pomiędzy wartościami granicznymi α (typowo wartość 1) a β (typowo wartość 3) – utrzymywanie małej liczby pakietów zbuforowanych ma na celu wykorzystanie skokowego przyrostu dostępnej przepływności. Algorytm TCP Vegas wprowadza więc inny sposób detekcji natłoku w sieci i przenosi ciężar działań z usuwania skutków natłoku (*congestion control*) na zapobieganie wystąpieniu natłoku (*congestion avoidance*). Algorytm ten zmienia w konsekwencji sposób reakcji na sygnały zbliżającego się stanu natłoku – maksymalny dozwolony rozmiar okna transmisji jest wyliczany bezpośrednio na podstawie pomiarów, nie zaś zwiększany i zmniejszany według z góry ustalonego schematu AIMD.

Poniższy pseudokod ilustruje sposób działania TCP Vegas. W tym algorytmie `cwnd` oznacza rozmiar okna natłoku, `ssthresh` oznacza poziom graniczny dla stanu *slow-start* (wielkości wyrażone w oktetach), `expected` i `actual` oznaczają odpowiednio oczekiwaną i rzeczywistą szybkość transmisji danych, a `base_rtt` i `rtt` wyrażają wyliczane i mierzone czasy RTT. Wielkość `seg_size` oznacza rozmiar segmentu danych i jest wartością stałą wyrażoną w oktetach, natomiast `alpha` i `beta` to wspomniane wcześniej parametry algorytmu o stałych wartościach. Vegas reaguje na trzy klasy zdarzeń: odebranie poprawnego potwierdzenia dostarczenia pakietu danych (ACK), utratę pakietu (3 powtórzone ACK lub upłynięcie granicznego czasu oczekiwania na potwierdzenie) oraz upłynięcie czasu jednego RTT. Jak łatwo zauważyć, modyfikacje szybkości transmisji odbywają się nie w takt zdarzeń otrzymywania potwierdzeń lub utraty pakietu, ale raz na każdy cykl RTT. Z podstawowego algorytmu Tahoe pozostawiono tu obsługę stanu *slow-start* i zdarzenia utraty pakietu. W pozostałych przypadkach szybkością transmisji steruje algorytm wyliczający tę wartość na podstawie pomiarów RTT.

```
INIT:
   cwnd := seg_size
   ssthresh ← default value > seg_size
   fast_recovery := false
   base_rtt ← RTT measured during TCP initiation
PERIODICALLY (once per RTT):
   base_rtt := min(base_rtt, rtt)
   expected := cwnd / base_rtt
   actual ← measured transmission rate
   if (expected - actual < alpha)
      cwnd := cwnd + seg_size /* linear increase per RTT */
   elsif (expected - actual > beta)
      cwnd := cwnd - seg_size /* linear dencrease per RTT */
   else /* alpha < expected - actual < beta */
      cwnd := cwnd /* no change */
```

```
   endif
ACK:
   if (cwnd <= ssthresh) /* slow-start phase */
      cwnd := cwnd + seg_size /* applicable during next RTT */
   endif
3 DUPACK or TIMEOUT: /* packet loss */
   ssthresh := cwnd/2
   cwnd := seg_size
```

TCP Westwood (Casetti, Gerla, Mascolo, Sansadidi, Wang, 2002), będący rozszerzeniem funkcjonalnym wariantu algorytmu TCP Reno, wprowadza funkcję rozróżniania utraty pakietu w sytuacji natłoku od utraty pakietu w wyniku zakłóceń transmisji. Działanie tego algorytmu opiera się o estymację dostępnego pasma transmisji w relacji między nadawcą a odbiorcą (ang. *end-to-end*) w celu ustalenia maksymalnego dozwolonego rozmiaru okna transmisji i poziomu granicznego dla stanu *slow-start*. Estymacja jest dokonywana po wejściu w stan natłoku identyfikowany jako trzy zduplikowane potwierdzenia lub upłynięcie granicznego czasu oczekiwania na potwierdzenie. Dzięki temu parametry szybkości transmisji ustalane są na „ostatnie bezpieczne" zamiast bezwarunkowego obniżania ich zgodnie z regułami AIMD. Dostępne pasmo jest określane przez dolnoprzepustowe filtrowanie chwilowych wartości strumienia pakietów potwierdzeń. Estymowana w ten sposób pożądana szybkość transmisji (ang. *eligible rate*) jest wykorzystywana także w czasie fazy adaptacyjnego próbkowania (ang. *agile probing*) będącej proponowaną modyfikacją zachowania w fazie *slow-start*. Dodatkowo, opracowany został schemat PNCD (ang. *persistent non-congestion detection*) dla wykrywania sytuacji trwałego braku natłoku pozwalających na przejście do fazy adaptacyjnego próbkowania zapewniającego lepsze wykorzystanie wzrostu dostępnej przepustowości sieci. TCP Westwood dokonuje więc adaptacyjnej modyfikacji maksymalnego dozwolonego rozmiaru okna transmisji i poziomu granicznego dla stanu *slow-start* biorąc pod uwagę wielkość przepustowości wykorzystywanej w chwili doznania stanu natłoku. Podobnie jak w przypadku TCP Vegas, algorytm TCP Westwood wylicza parametry transmisji na podstawie pomiarów, nie zaś według schematu AIMD.

Poniższy pseudokod ilustruje sposób działania TCP Westwood. W tym algorytmie `cwnd` oznacza rozmiar okna natłoku, `ssthresh` oznacza poziom graniczny dla stanu *slow-start* (wielkości wyrażone w oktetach), , a `rtt_min` i `rtt` wyrażają wyliczane i mierzone czasy RTT. Wielkość `seg_size` oznacza rozmiar segmentu danych i jest wartością stałą wyrażoną w oktetach, natomiast `BWE` to wartość estymowanej przepustowości wyliczanej zgodnie ze wspomnianym wcześniej algorytmem. Westwood reaguje na cztery klasy zdarzeń: odebranie potwierdzenia dostarczenia pakietu danych (ACK), odebranie powtórzonego (DUPACK), utraty pakietu (upłynięcie granicznego czasu oczekiwania na potwierdzenic) oraz upłynięcie cza-

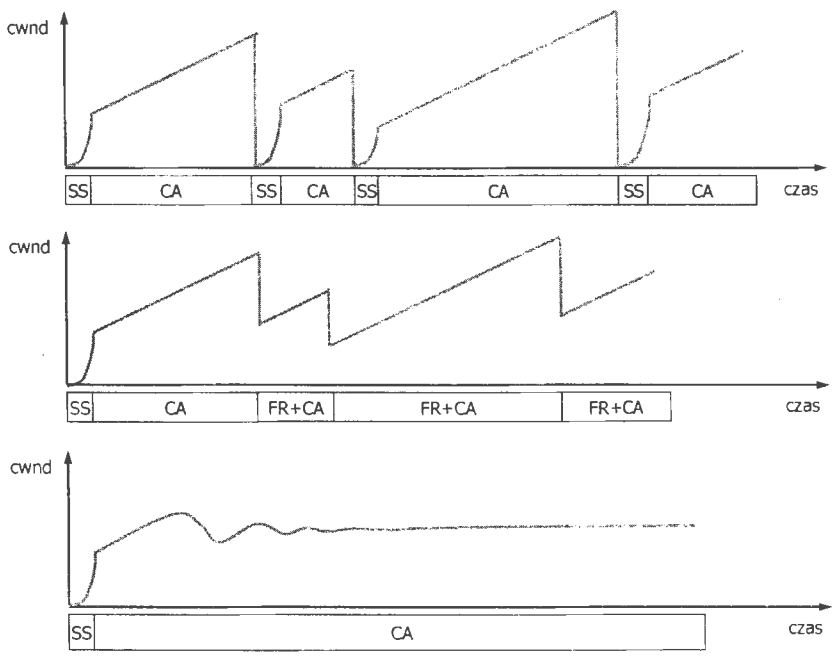
su jednego RTT. Jak łatwo zauważyć, modyfikacje szybkości transmisji odbywają się w takt zdarzeń otrzymywania potwierdzeń lub utraty pakietu, ale raz wartości wyznaczane są przez estymowaną dostępną przepustowość. Z podstawowego algorytmu Tahoe pozostawiono tu obsługę stanu *slow-start*. W pozostałych przypadkach szybkością transmisji steruje algorytm wyliczający tę wartość na podstawie estymowanej przepustowości.

```
INIT:
   cwnd := seg_size
   ssthresh ← default value > seg_size
   fast_recovery := false
PERIODICALLY (once per RTT):
   BWE ← estimated according to eligible rate method
   rtt_min := min(rtt_min, rtt)
ACK:
   if (cwnd <= ssthresh) /* slow-start phase */
      cwnd := cwnd + seg_size
   else /* congestion avoidance phase */
      cwnd := cwnd + 1/cwnd
   endif
n DUPACK:
   ssthresh := BWE * rtt_min
   if (cwnd > ssthresh)
      cwnd := ssthresh
   endif
TIMEOUT: /* packet loss */
   ssthresh := BWE * rtt_min
   if (ssthresh < 2 * seg_size)
      ssthresh := 2 * seg_size
   endif
   cwnd := seg_size
```

Rysunek przedstawiony na następnej stronie podsumowuje porównanie charakterystyki działania algorytmów przez przedstawienie hipotetycznego przebiegu w czasie szybkości transmisji dla algorytmów działających w tych samych warunkach pracy i reagujący na te same zdarzenia zachodzące w czasie realizacji transmisji danych.

5. Wnioski i perspektywy badawcze

Dzięki wprowadzeniu współpracy mechanizmów kontroli przepływu i zarządzania natłokiem, podstawowe funkcje protokołu TCP działają na zasadach adaptacyjnych, objawiających się modyfikacją chwilowej szybkości transmisji danych w odpowiedzi na sygnały o stanie natłoku. Ta podstawowa adaptacja w zachowaniu nadawcy jest realizowana w oparciu o statyczny schemat AIMD lub przez wprowadzenie dodatkowych parametrów sterujących.



Rys. 2. Algorytmy zarządzania natłokiem w działaniu

W wyniku prac prowadzonych w zakresie wzmocnienia cech adaptacyjności algorytmów TCP zaproponowane zostały modyfikacje podstawowych algorytmów zarządzania natłokiem (TCP Reno, TCP Vegas) nadające im nowy wymiar adaptacyjności – TCP-Adaptive Reno (Shimonishi, Hama, Murase, 2007) czy TCP New-Vegas (De Vendictis, Baiocchi, Bonacci, 2003). Pomysły te bazują na wcześniej opracowanych metodach estymacji dostępnego pasma i włączeniu dodatkowej „warstwy" adaptacji uzależniającej parametry algorytmów od wyników estymacji.

Obszar adaptacji w zarządzaniu natłokiem protokołu TCP nie został jednak systematycznie zbadany. W tym obszarze należy poszukiwać dodatkowych możli-

wości zaawansowanej adaptacji w zarządzaniu natłokiem – zarówno wpływania na istniejące parametry algorytmów, jak i wykorzystania innych metod estymacji stanu sieci, do których zarządzanie natłokiem powinno się adaptować.

Literatura

[1]. Miller G., Thompson K. (1998): *The nature of the beast: recent traffic measurements from an Internet backbone*, CAIDA/University of California, http://www.caida.org/publications/papers/1998/Inet98/Inet98.html

[2]. Leinen S., Schulzrinne H. (2001): *Internet traffic measurements – Long-Term Traffic Statistics*, Columbia University, http://www.cs.columbia.edu/~hgs/internet /traffic.html

[3]. Floyd S., Jacobson V. (1993): Random early detection gateways for congestion avoidance, *IEEE/ACM Trans. Networking*, vol. 1, pp. 397-413, ftp://ftp.ee.lbl.gov/papers/early.ps.gz

[4]. K. Ramakrishnan, S. Floyd, D. Black (2001): *The Addition of Explicit Congestion Notification (ECN) to IP*, http://www.ietf.org/rfc/rfc3168.txt

[5]. Athuraliya S., Li V.H., Law S.H., Yin Q. (2001): REM: Active queue management, *IEEE Network*, vol. 15, pp. 48-53, extended version Proc. ITC17, Salvador, Brazil, http://netlab.caltech.edu/

[6]. Kelly F.P., Maulloo A., Tan D. (1998): Rate control for communication networks: Shadow prices, proportional fairness and stability, *J. Oper. Res. Soc.*, vol. 49, no. 3, pp. 237-252

[7]. Low S., Paganini F., Doyle J. (2002): Internet Congestion Control, *IEEE Control Systems Magazine*

[8]. Jacobson V. (1988): *Congestion avoidance and control*, Proc. SIGCOMM '88, ACM, ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z

[9]. Alman M., Paxson V., Stevens W. (1999): TCP Congestion Control, RFC 2581, IETF

[10]. Brakmo L., Peterson L. (1995): TCP Vegas: End to end congestion avoidance on a global Internet, *IEEE J. Select. Areas Commun.*, vol. 13, pp. 1465-1480, http://cs.princeton.edu/nsg/papers/jsac-vegas.ps

[11]. Mascolo S., Casetti C., Gerla M., Sanadidi M. Y., Wang R. (2001): *TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links*, Proc. 7th International Conference on Mobile Computing and Networking, pp. 287-297.

[12]. Casetti C., Gerla M., Mascolo S., Sansadidi M., Wang R. (2002): TCP Westwood: end-to-end congestion control for wired/wireless networks, *Wireless Network Journal*, vol. 8, pp. 467-479.

[13]. Shimonishi H., Hama T., and Murase T. (2007): *TCP-Adaptive Reno for Improving Efficiency-Friendliness Tradeoffs of TCP Congestion Control Algorithm.*

[14]. De Vendictis A., Baiocchi A., Bonacci M. (2003): *Analysis and enhancement of TCP Vegas congestion control in a mixed TCP Vegas and TCP Reno network scenario*, Elsevier Science B.V.

ISBN 9788389475220