УДК 004.05

Струбицький Р.П.

РОЗРОБКА ПРОТОКОЛУ СЕАНСОВОГО РІВНЯ ДЛЯ ВИСОКОШВИДКІСНИХ РЕГІОНАЛЬНО-РОЗПОДІЛЕНИХ МЕРЕЖ

Національний університет «Львівська політехніка»

rolleks@gmail.com

Представлено реалізацію протоколу на базі UDP, оптимізовану за ефективність передачі даних. Приведено результати практичного тестування протоколу, продемонстровано можливість реалізації ефективних та практичних додатків на базі протоколів UDP

Ключові слова: UDP, протокол, сеансовий рівень

Вступ

Швидке збільшення пропускної здатності комп'ютерних мереж дозволило розробляти численні додатки, які передбачають інтенсивну обробку даних. Ці нові додатки можуть виконувати задачі, починаючи від масової передачі даних (наприклад, *SDSS* [20] та *e-VLBI*), до інтерактивних систем високої пропускної здатності (наприклад, *GeoWall* [14]).

Актуальність дослідження

Різні програми ставлять різні вимоги до послуг передачі даних. Наприклад, додаток *GeoWall* може віддати перевагу плавній зміні швидкості передачі даних, тоді як для додатку *SDSS* бажано, щоб дані передавалися з максимально можливою швидкістю в приватних мережах.

Проте, нинішня система Інтернет призначена для забезпечення підтримки цілої множини різного типу додатків. Ця філософія дизайну Інтернету має великий вплив на розвиток транспортних протоколів. Більшість трафіку в Інтернеті формується за рахунок потоків *TCP*, але існують додатки для яких протокол *TCP* не забезпечує достатнього рівня ефективності. У контексті високопродуктивних обчислень, *TCP* добре відомий своєю низькою ефективністю та справедливим поділом ресурсів в мережах з високою затримкою пропускної здатності [5, 13].

Аналіз останніх досліджень

За останні декілька років дослідники комп'ютерних мереж запропонували багато нових алгоритмів керування перевантаженням для загальної та специфічної передачі даних [6, 17]. Тим не менше, більшість з цих запропонованих алгоритмів завершуються моделюванням та обмежуються умовами лабораторних експериментів; мало з них пройшли практичне тестування і випробування в реальних високопродуктивних мережах. З іншого боку, модифікації мережевого стеку ядра протоколу (наприклад, нові варіанти *TCP*) зазвичай вимагають кількох років для стандартизації, впровадження та широкого розгортання. Справді, з часів появи протоколу *TCP*, близько трьох десятиліть тому, тільки його чотири версії були широко розгорнуті, а саме *Tahoe, Reno, NewReno,* і *SACK* [5, 16, 21].

Хоча на сьогоднішній день все більше мереж отримують швидкість передачі даних 1 Гбіт/с і вище, але як і раніше актуальною проблемою для грід-додатків залишається використання широкої смуги пропускної здатності, через обмеження існуючих транспортних протоколів мережі [21]. Обмеження діючих мережевих транспортних протоколів є однією з головних причин, через яку так важко масштабувати додатки з інтенсивним використанням мережевих з'єднань від місцевих кластерів до глобальних мереж [1, 3, 7, 21].

Протокол управління передачею (*TCP*) успішно використовується протягом десятиліть, як основний протокол транспортного рівня стеку мережних протоколів. Проте останнім часом було пока-

зано, що *TCP* має деякі втрати продуктивності при його використанні для високошвидкісних мереж *Wide Area*, особливо для географічно віддалених мереж. Алгоритм управління перевантаженням *AIMD*, який використовується протоколом *TCP*, є досить "бідним" у розкритті доступної смуги пропусканої здатності і у випадку великої втрати пакетів у високопродуктивних мережах з досить великими часами затримки [5].

Дослідники комп'ютерних мереж працюють над новими транспортними протоколами і алгоритмами контролю насичення для підтримки високошвидкісних мереж наступного покоління. Багато робіт, у тому числі варіантів *TCP* (*FAST* [4], *BiC* [6], *Scalable* [16], і *HighSpeed* [23]) і *XCP* [15] показали більш високу продуктивність при їх моделюванні. Однак практичне використання в реальних додатках цих протоколів все ще дуже обмежена через труднощі їх реалізації, встановлення та обмеження технічного рівня. Користувачі мережі, яким потрібно передавати великі масиви даних (наприклад, в грідкомп'ютингу) зазвичай звертаються до рішень рівня додатків, серед яких дуже популярні протоколи на основі *UDP*, наприклад *SABUL* [9], *UDT* [13], *Tsunami* [18], *RBUDP* [19], *FOBS* [2] і *GTP* [22].

Протоколи на основі *UDP* забезпечують набагато кращу переносимість і прості в при їх встановленні. Однак, незважаючи на простоту впровадження протоколів на рівні користувача, досить важко налаштувати їх в ядрі, щоб зробити їх максимально ефективними. Оскільки реалізації рівня користувач не можуть змінити код ядра, можуть бути додаткові перемикання контексту і копіюванням ділянок пам'яті між рівнем користувача та рівнем ядра. На високих швидкостях передачі даних, ці операції дуже чутливі до завантаження процесора і продуктивності протоколу.

Виділення невирішених проблем

Враховуючи перечисленні затруднення та недоліки існуючого стану передачі даних великого обсягу через високопродуктивні регіонально-розподілені мережі видається доцільним з практичної точки зору розробка протоколу сеансового рівня для таких мереж.

Новий протокол сеансового рівня повинен бути сумісним з стандартними протоколами стеку протоколів *OSI-TCP* та *UDP*. Крім того, він повинен бути адаптований для сучасних мереж, тобто ефективно використовувати пропускну здатність мережі, не залежно від її характеристик. При використанні такого протоколу не повинні збільшуватися затримки при переході передачі між різними типами мереж. Тим самим протокол повинен забезпечити ефективне використання ресурсів крайніх вузлів мережі. У питанні захищенності - він повинен підтримувати шифрування даних, тобто він повинен дозволяти підключення протоколів шифрування.

Виклад основного матеріалу

У роботі представлено оптимізацію ефективності реалізації протоколу на базі *UDP* і показано, що за цими ідеями можна реалізувати ефективні та практичні додатки на базі протоколів *UDP*. Наприклад, використання середовища протоколу *UDT* (базований на *UDP*) [13], може легко підтримувати різні алгоритми управління перевантаженням, наприклад, високошвидкісні *TCP* [4, 6, 16, 23] або вибуховий *RBUDP* [23].

Перевагами даного середовища протоколу є:

1. Нові протоколи на базі *UDP* можуть бути швидко прототиповані і випробувані;
2. Різноманітні алгоритми управління перевантаженням можна легко порівняти експериментально;
3. Конкретні прикладні протоколи, що використовують *UDP*, можуть бути розроблені відносно легко.

Також, з використанням даного середовища можуть розроблятися конкретні протоколи для розподілених даних або потокових медіа-додатків.

Основним недоліком даного середо-

вища протоколу є додаткове навантаження самого середовища, які накладають додаткові витрати протоколу прикладного рівня у порівнянні з рівнем протоколу ядра.

Для досягнення високої продуктивності, керуються двома принципами:

1. Не повинно бути піків завантаження процесора, особливо на стороні отримання даних. Сплески завантаження *CPU* можуть призвести до невчасної обробки прийнятих пакетів, що у свою чергу приведе до втрати даних передачі. Це може привести до серйозних проблем, які спостерігаються в *TCP* з *AIMD*, у якому при втраті пакету, зазвичай, витрачається багато часу для відновлення передачі даних при з'єднаннях на великих відстанях.

2. Загальне завантаження процесора повинно бути якомога меншим. Проста реалізація додатку може запобігти надмірному використанню центрального процесора. Крім того, інші операції з обробки даних, які також потребують процесорного часу, часто виконуються паралельно з передачею даних.

Для перевірки завантаженості центрального процесора (*CPU*), системних затримок та впливу розмірів пакетів та розмірів буферів на швидкість передачі даних для протоколу *UDP* було проведено декілька практичних експериментів.

Експерименти проводилися на п'яти різних системах. Усі вони були з'єднані з іншим віддаленим вузлом з аналогічною конфігурацією, через мережу з пропускною здатністю не менше ніж швидкість *NIC*. *MTU* на стендах становила 1500 байт.

При використанні розміру пакету *UDP* 1500 байт і розміру буфера *UDP* 1 МБ були отримані результати наведені у таблиці 2. Використання процесора вимірювалося в одиницях МГц/Мбіт, що є відношенням частоти використання процесора до швидкості передачі даних. Експеримент проводився в локальних мережах. (Частота процесора визначалась, як сума частот всіх процесорів, які використовувалися додатком при тестуванні. У деяких системах кількість процесорів була менше 0,5, в яких використовувалася технологія *Hyper-Threading*.)

Таблиця 1. Середовища експерименту

Ім'я	*CPU*	*Memory*	*NIC*	*OS*
onno	*Dual Itanium*2 1.5*GHz*	8 *GB*	10 *GbE*	*Linux* 2.6.0
*sara*77	*Dual Xeon* 2.4*GHz*	2 *GB*	1 *GbE*	*Linux* 2.4.18
*ncdm*171	*Dual PowerPC G*4 1*GHz*	2 *GB*	1 *GbE*	*Mac OS X*
*win*91	*Dual Xeon* 2.4*GHz*	2 *GB*	1 *GbE*	*Windows* 7
*ncdm*87	*Dual Opteron* 2.4*GHz*	4 *GB*	1 *GbE*	*Linux* 2.6.8

Таблиця 2. Використання центрального процесора і час затримки для *UDP* та *TCP*

Ім'я	*UDP*			*TCP*		
	CPU Util. (MHz/Mbps)		*Delay* (*ms*)	*CPU Util. (MHz/Mbps)*		*Delay* (*ms*)
	Відправка	Отримання		Відправка	Отримання	
onno	0.22	0.35	0.062	0.23	0.50	0.068
*sara*77	0.40	0.45	0.070	0.51	0.51	0.086
*ncdm*171	1.22	1.45	0.202	2.22	2.73	0.245
*win*91	1.03	1.09	0.203	1.14	1.28	0.302
*ncdm*87	0.26	0.40	0.065	0.25	0.56	0.087

Для усунення впливу помилок *RTT* на результат, було використано локальні мережеві з'єднання з дуже малим значеннях *RTT*, тому що, в іншому випадку, великі значення *RTT* можуть порушити точність моделювання.

Результати експериментальних досліджень є усередненням результатів 100 випробувань.

Табл. 2 показує, що протокол *UDP* створює менше завантаження центрального процесора і дає менші часи затрим-

ки, ніж *TCP*, в основному за рахунок того, що цей протокол має менші накладні витрати на обробку даних, які передаються протоколом. Це означає, що ефективна реалізація протоколу на основі *UDP* може мати таку ж продуктивність, як і вбудована в ядро системи реалізація *TCP*.

Для визначення факторів впливу на продуктивність протоколу було проведено ряд додаткових експериментів зі зміною двох параметрів:

1. Розмір буфера *UDP*;
2. Розмір пакету протоколу.

Результати порівняння між двома локальними машинами зображено на рисунках 1 і 2.

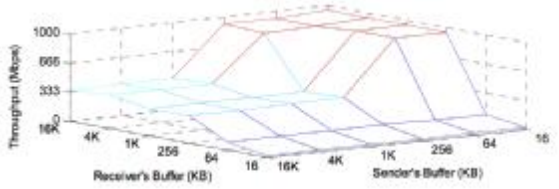

Рис. 1. Залежність продуктивності *UDP* від розміру буфера

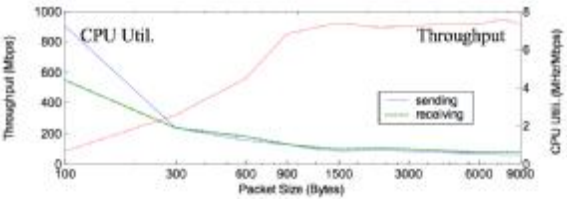

Рисунок 2. Залежність продуктивності *UDP* від розміру пакету

Рис. 1 демонструє, що розмір буфера на стороні отримувача є вирішальним фактором пропускної здатності. Розмір буфера повинен бути достатньо великим для досягнення оптимальної пропускної здатності. Графік залежності пропускної здатності від розміру буфера (Рис. 2) демонструє, що є деяке оптимальне значення розміру буфера, і подальше збільшення його розмірів не дають приросту пропускної здатності, а навпаки, відбувається зниження. Практично ж, розмір буферу на стороні передавача може бути значно меншим, ніж на стороні отримувача тому що:

1. Великий буфер збільшує *RTT*;
2. Великий буфер збільшує можливе перевантаження мережі.

Перша ситуація викликає більш серйозні проблеми, коли відбувається втрата пакетів, в той час, як друга викликає більше втратів пакетів. Розмір буфера на стороні відправника повинен бути достатньо великим, щоб не було обмежень на швидкість передачі пакетів. Це мінімальне значення зв'язане зі значенням *RTT*.

Залежність на рис. 2 демонструє, що розмір пакету повинен бути рівний розміру *MTU*. Хоча видно, що більший розмір пакетів приводить до меншої завантаженості процесора (оскільки є менше накладних витрат на обробку).

Також, підвищення продуктивності забезпечується зменшенням розмірів пакетів. Цей метод може бути використаний, щоб уникнути додаткового копіювання пам'яті, уникаючи використання тимчасового буферу для упаковки і розпаковування пакетів і даних користувача.

На відміну від стандартних протоколів транспортного рівня *TCP* і *UDP*, протокол *UDT* не входить до стеку протоколів *OSI* і тому ще не є стандартом дефакто для комп'ютерних мереж. Це обмеження потребує додаткової програмної реалізації цього протоколу. Його базування на стандартних протоколах *TCP* і *UDP* дещо спрощує цю задачу.

Протокол *UDT* - надійний транспортний протокол передачі потокових даних орієнтований на з'єднання на рівні додатків, який реалізований мовою програмування *C++*. Архітектура протоколу зображена на рисунку 3. Протокол *UDT* містить п'ять функціональних компонентів: модуль *API*, відправник, одержувач, слухач, а також канал *UDP*, дана функціональна структура представлена на рис. 4. Протокол *UDT* також включає в себе чотири компоненти даних: буфер протоколу відправника, буфер протоколу одержувача, список втрат відправника і список втрат одержувача. Протокол *UDT* є двостороннім промисловим протоколом, усі його елементи мають однакову структуру [8].

При роботі з протоколом користувач використовує два типи пакетів: пакети даних і пакети управління. Вони відріз-

няються першим бітом в заголовку пакету. Пакет даних *UDT* включає в себе пакетний порядковий номер, порядковий номер повідомлення, і штамп часу з моменту ініціювання з'єднання [11].

Рис. 3. Архітектура *UDT*

Модуль *API* відповідає за зв'язок з додатками. Дані для відправки передаються в буфері відправника і передається відправником у канал *UDP*. Приймач зчитує дані з каналу *UDP* і розміщує їх в буфері приймача, сортує дані і перевіряє втрати пакетів. Додатки можуть читати отримані дані з буфера приймача. Приймач також зберігає отриману інформацію управління. Він оновлює список втрат відправника, коли *NAK* прийнято і список втрати приймача при виявленні втрати. Деякі події управління ініціюють приймач оновити модуль управління перевантаженням [10].

Протокол *UDT* використовує керуванням перевантаженням на основі зміни швидкості та зміною розмірів вікна на основі управляння вихідним трафіком. Управління швидкістю оновлює період відправки пакету кожен інтервал часу, в той час, як управління потоком оновлює розмір вікна щоразу після отримання підтвердження. Протокол *UDT* використовує таймер на основі підтверджень (*AK*), створює підтвердження з інформацією про те, чи є нові безперервно отримані пакети даних. *UDT* відправляє набір пакетів даних з фіксованим розміром пакетів, якщо немає достатніх даних для відправки. Фіксований розмір можна налаштувати додатково і оптимальне значення є максимальний розмір блоку (*MTU*) [12].

Рис. 4. Функціональна структура протоколу *UDT*

Для практичного використання протоколу транспортного рівня *UDP* на сеансовому рівні доречно створити надбудову, яка структуризує інформаційні потоки і спрощує використання даного протоколу в практичних додатках.

Надбудова над протоколом *UDT*, що фактично стає протоколом сеансового рівня, розширює функціонал протоколу, забезпечуючи користувачу використання не тільки передачі даних в сирому вигляді, але й для передачі файлових структур даних.

Заголовок пакету протоколу має наступну структуру (Рисунок 5):

1 байт - номер версії протоколу (для можливості модифікації протоколу, та зворонтьої сумістності);

1 байт - ідентифікатор команди яка передається;

2 байти - ідентифікатор каналу (для мультиплексування);

3 байти - кількість 16-байтних блоків у пакеті;

1 байт - довжина додаткового заповнення пакету (для забезпечення кратності довжині ключа шифрування);

8 байт - службова інформація пакету даних.

Рис. 5. Структура заголовку пакету даних протоколу

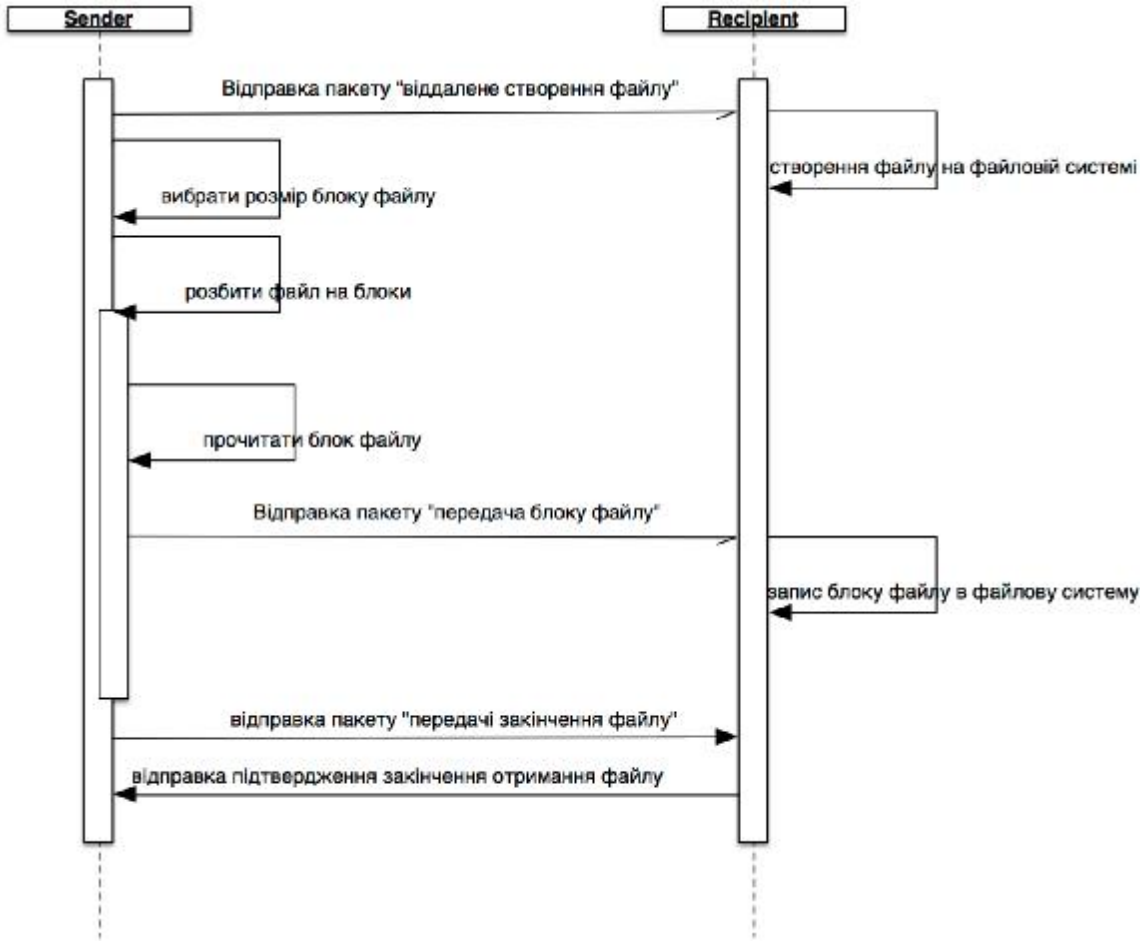

Рис. 6. Діаграма послідовності передачі даних протоколом

Для забезпечення можливості шифрування даних, довжина пакету повинна бути кратна довжині ключа шифрування. Дана вимога не є вимогою протоколу передачі - це вимоги криптографічних алгоритмів, а тому приймається аксіоматично.

Для забезпечення повноти функціоналу протоколу сеансового рівня вводяться наступні типи команд, що повністю відсутні в первинній версії протоколу:

- віддалене створення директорії;
- віддалене створення файлу;
- передача блоку файлу;
- передача закінчення файлу;

Ефективність використання ресурсів вузлів передачі та отримання досягається передачею блоків файлу. Розмір блоків повинен бути кратний розміру буфера файлової системи.

Згідно з запропонованою надбудовою над транспортним протоколом і рекомендаціями щодо практичного використання, передача одного файлу (при встановленому з'єднанні) проводиться за наступним алгоритмом (Рис. 6):

1. відправка пакету "віддалене створення файлу".
2. передача змісту файлу:
   a. вибір розмір блоку файлу.
   b. поділ файлу на блоки.
   c. послідовна відправка пакету "передача блоку файлу" для кожного блоку.
3. відправка пакету "передача закінчення файлу".
4. очікування підтвердження закінчення отримання файлу.

З приймаючої сторони алгоритм роботи наступний (при встановленому з'єднанні):

1. отримання пакету "віддалене створення файлу".
2. створення файлу на файловій системі.
3. отримання змісту файлу:
   a. отримання пакету "передача блоку файлу" по блочно.
   b. запис блоку файлу в файлову систему.
4. отримання пакету "передача закінчення файлу".
5. закриття файлу.
6. відправка підтвердження закінчення отримання файлу.

Висновки

Використання даної технології дає можливість оптимальної та ефективної реалізації протоколу на базі *UDP*, для передачі команд файлової системи в хмарковому сховищі даних. На базі чого можна реалізувати ефективні та практичні додатки.

Перевагою даного протоколу є:

- вбудовані команди файлової системи
- використання високорівневих команд
- можливість мультиплексування різних даних по одному каналу зв'язку
- можливість передачі одного файлу по багатьох каналах зв'язку одночасно

Список літератури

1. Data management and transfer in high-performance computational grid environments / Bill Allcock, Joe Bester, John Bresnahan et al. // Parallel Computing. – 2002. – Vol. 28, no. 5. – Pp. 749–771.

2. Dickens Phillip M. FOBS: A lightweight communication protocol for grid computing // Euro-Par 2003 Parallel Prcessing. – Springer, 2003. – Pp. 938–946.

3. Experimental studies using photonic data services at IGrid 2002 / Robert L Grossman, Yunhong Gu, Don Hamelburg et al. // Future Generation Computer Systems. – 2003. – Vol. 19, no. 6. – Pp. 945–955.

4. FAST TCP: motivation, architecture, algorithms, performance / David X Wei, Cheng Jin, Steven H Low, Sanjay Hegde // IEEE/ACM Transactions on Networking (ToN). – 2006. – Vol. 14, no. 6. – Pp. 1246–1259.

5. Feng Wu-chun, Tinnakornsrisuphap Peerapol. The failure of TCP in high-performance computational grids // Supercomputing, ACM/IEEE 2000 Conference / IEEE. – 2000. – Pp. 37–37.

6. Floyd Sally. HighSpeed TCP for large congestion windows. – 2003.

7. Foster Ian, Kesselman Carl, Tuecke Steven. The anatomy of the grid: Enabling scalable virtual organizations // International journal of high performance computing applications. – 2001. – Vol. 15, no. 3. – Pp. 200–222.

8. Gu Yunhong. UDT: a high performance data transport protocol. – University of Illinois at Chicago, 2005.

9. Gu Yunhong, Grossman Robert. SABUL: A transport protocol for grid computing // Journal of Grid Computing. — 2003. – Vol. 1, no. 4. – Pp. 377–386.

10. Gu Yunhong, Grossman Robert L. End-to-end congestion control for high performance data transfer // submitted to IEEE/ACM Transaction on Networking. – 2003.

11. Gu Yunhong, Grossman Robert L. Supporting configurable congestion control in data transport services // Proceedings of the 2005 ACM/IEEE conference on Supercomputing / IEEE Computer Society. – 2005. – P. 31.

12. Gu Yunhong, Grossman Robert L. UDT: UDP-based data transfer for high-speed wide area networks // Computer Networks. – 2007. – Vol. 51, no. 7. – Pp. 1777–1799.

13. Gu Yunhong, Hong Xinwei, Grossman Robert L. Experiences in design and implementation of a high performance transport protocol // Supercomputing, 2004. Proceedings of the ACM/IEEE SC2004 Conference / IEEE. – 2004. – Pp. 22–22.

14. JuxtaView-a tool for interactive visualization of large imagery on scalable tiled displays / Naveen K Krishnaprasad, Venkatram Vishwanath, Shalini Venkataraman et al. // Cluster Computing, 2004 IEEE International Conference on / IEEE. – 2004. – Pp. 411–420.

15. Katabi Dina, Handley Mark, Rohrs Charlie. Congestion control for high bandwidth-delay product networks // ACM SIGCOMM Computer Communication Review. – 2002. – Vol. 32, no. 4. – Pp. 89–102. 16. Kelly Tom. Scalable TCP: Improving performance in highspeed wide area networks // ACM SIGCOMM Computer Communication Review. – 2003. – Vol. 33, no. 2. – Pp. 83–91.

17. Kurose James F. Computer networking: a top-down approach featuring the Internet. – Pearson Education India, 2005. 18. Meiss Mark R et al. Tsunami: A high-speed rate-controlled protocol for file transfer. – 2004.

19. Reliable blast UDP: Predictable high performance bulk data transfer / Eric He, Jason Leigh, Oliver Yu, Thomas A DeFanti // Cluster Computing, 2002. Proceedings. 2002 IEEE International Conference on / IEEE. – 2002. – Pp. 317–324. 20. The SDSS skyserver: public access to the sloan digital sky server data / Alexander S Szalay, Jim Gray, Ani R Thakar et al. // Proceedings of the 2002 ACM SIGMOD international conference on Management of data / ACM. – 2002. – Pp. 570–581.

21. Transport protocols for high performance: Whither TCP / A Chien, T Faber, A Falk et al. // Communications of the ACM. – 2003. – Vol. 46, no. 11. – Pp. 42–49.

22. Wu XINRAN, Chien Andrew. Evaluation of rate-based transport protocols for lambda-grids // High performance Distributed Computing, 2004. Proceedings. 13th IEEE International Symposium on / IEEE. – 2004. – Pp. 87–96.

23. Xu Lisong, Harfoush Khaled, Rhee Injong. Binary increase congestion control (BIC) for fast long-distance neworks // INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies / IEEE. – Vol. 4. – 2004. – Pp. 2514– 2524.

Статтю подано до редакції 22.04.2015