



US008805948B2

(12) **United States Patent**
Boucher et al.

(10) **Patent No.:** **US 8,805,948 B2**
(45) **Date of Patent:** **Aug. 12, 2014**

(54) **INTELLIGENT NETWORK INTERFACE
SYSTEM AND METHOD FOR PROTOCOL
PROCESSING**

(71) Applicant: **Alacritech, Inc.**, San Jose, CA (US)

(72) Inventors: **Laurence B. Boucher**, Saratoga, CA
(US); **Stephen E. J. Blightman**, San
Jose, CA (US); **Peter K. Craft**, San
Francisco, CA (US); **David A. Higgen**,
Apopka, FL (US); **Clive M. Philbrick**,
San Jose, CA (US); **Daryl D. Starr**,
Milpitas, CA (US)

(73) Assignee: **A-Tech LLC**, Newark, DE (US)

(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 0 days.

(21) Appl. No.: **14/038,297**

(22) Filed: **Sep. 26, 2013**

(65) **Prior Publication Data**

US 2014/0059155 A1 Feb. 27, 2014

Related U.S. Application Data

(63) Continuation of application No. 09/692,561, filed on
Oct. 18, 2000, now Pat. No. 8,631,140, which is a
continuation of application No. 09/067,544, filed on
Apr. 27, 1998, now Pat. No. 6,226,680.

(60) Provisional application No. 60/061,809, filed on Oct.
14, 1997.

(51) **Int. Cl.**

G06F 15/16 (2006.01)
H04L 29/06 (2006.01)
H04L 29/08 (2006.01)
H04L 29/12 (2006.01)
H04L 12/56 (2006.01)
H04Q 3/00 (2006.01)
G06F 5/10 (2006.01)

(52) **U.S. Cl.**

CPC **H04L 61/25** (2013.01); **H04L 69/168**
(2013.01); **H04L 69/08** (2013.01); **H04Q**
2213/13204 (2013.01); **H04L 69/16** (2013.01);
H04L 29/06 (2013.01); **H04L 69/22** (2013.01);
H04L 67/34 (2013.01); **H04Q 2213/1332**
(2013.01); **H04L 29/12009** (2013.01); **H04L**
69/166 (2013.01); **H04L 45/245** (2013.01);
H04L 67/325 (2013.01); **H04L 67/10**
(2013.01); **H04L 69/163** (2013.01); **H04L**
45/00 (2013.01); **H04L 69/165** (2013.01);
H04Q 3/0029 (2013.01); **H04L 69/12**
(2013.01); **H04L 69/161** (2013.01); **H04L**
61/10 (2013.01); **H04L 49/9094** (2013.01);
H04L 67/327 (2013.01); **H04L 29/12018**
(2013.01); **H04L 49/9063** (2013.01); **H04Q**
2213/13103 (2013.01); **G06F 5/10** (2013.01);
H04L 69/162 (2013.01); **H04Q 2213/13345**
(2013.01); **H04L 69/32** (2013.01); **H04Q**
2213/13093 (2013.01); **H04L 49/90** (2013.01);
H04L 49/901 (2013.01); **H04Q 2213/13299**
(2013.01)

USPC **709/212**; 709/224; 370/474

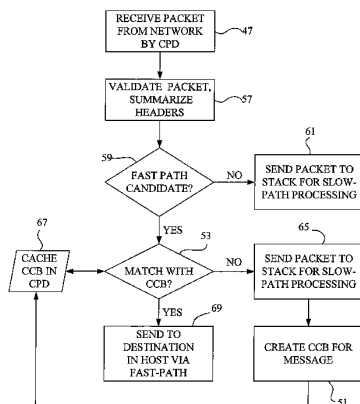
(58) **Field of Classification Search**

CPC H04L 69/19; H04L 69/161; H04L 69/163;
H04L 69/165; H04L 69/168
USPC 709/200–202, 212, 224, 230, 236, 238;
370/389, 392, 419, 463, 469, 479
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,366,538 A	12/1982	Johnson et al.
4,485,455 A	11/1984	Boone et al.
4,485,460 A	11/1984	Stambaugh
4,589,063 A	5/1986	Shah et al.
4,700,185 A	10/1987	Balph et al.
4,991,133 A	2/1991	Davis et al.
5,056,058 A	10/1991	Hirata et al.
5,058,110 A	10/1991	Beach et al.
5,097,442 A	3/1992	Ward et al.
5,129,093 A	7/1992	Muramatsu et al.
5,163,131 A	11/1992	Row et al.
5,212,778 A	5/1993	Dally et al.



US 8,805,948 B2

Page 2

5,274,768	A	12/1993	Traw et al.	5,926,642	A	7/1999	Favor
5,280,477	A	1/1994	Trapp	5,930,830	A	7/1999	Mendelson et al.
5,281,963	A	1/1994	Ishikawa et al.	5,931,918	A	8/1999	Row et al.
5,289,580	A	2/1994	Latif et al.	5,935,205	A	8/1999	Murayama et al.
5,303,344	A	4/1994	Yokoyama et al.	5,935,249	A	8/1999	Stern et al.
5,412,782	A	5/1995	Hausman et al.	5,937,169	A	8/1999	Connery et al.
5,418,912	A	5/1995	Christenson	5,941,969	A	8/1999	Ram et al.
5,448,566	A	9/1995	Richter et al.	5,941,972	A	8/1999	Hoese et al.
5,485,455	A	1/1996	Dobbins et al.	5,950,203	A	9/1999	Stakuis et al.
5,485,460	A	1/1996	Schrier et al.	5,963,876	A	10/1999	Manssen et al.
5,485,579	A	1/1996	Hitz et al.	5,970,804	A	10/1999	Robbat, Jr.
5,506,966	A	4/1996	Ban	5,978,844	A	11/1999	Tsuchiya et al.
5,511,169	A	4/1996	Suda	5,987,022	A	11/1999	Geiger et al.
5,517,668	A	5/1996	Szwerinski et al.	5,991,299	A	11/1999	Radogna et al.
5,524,250	A	6/1996	Chesson et al.	5,996,013	A	11/1999	Delp et al.
5,535,375	A	7/1996	Eshel et al.	5,996,024	A	11/1999	Blumenau
5,548,730	A	8/1996	Young et al.	6,005,849	A	12/1999	Roach et al.
5,553,241	A	9/1996	Shirakhar	6,009,478	A	12/1999	Panner et al.
5,566,170	A	10/1996	Bakke et al.	6,014,380	A	1/2000	Hendel et al.
5,574,919	A	11/1996	Netravali et al.	6,014,557	A	1/2000	Morton et al.
5,588,121	A	12/1996	Reddin et al.	6,016,513	A	1/2000	Lowe
5,590,328	A	12/1996	Seno et al.	6,021,446	A	2/2000	Gentry et al.
5,592,622	A	1/1997	Isfeld et al.	6,021,507	A	2/2000	Chen
5,596,574	A	1/1997	Perlman et al.	6,026,452	A	2/2000	Pitts
5,598,410	A	1/1997	Stone	6,034,962	A	3/2000	Ohno et al.
5,619,650	A	4/1997	Bach et al.	6,034,963	A	3/2000	Minami et al.
5,629,933	A	5/1997	Delp et al.	6,038,562	A	3/2000	Anjur et al.
5,633,780	A	5/1997	Cronin	6,041,058	A	3/2000	Flanders et al.
5,634,099	A	5/1997	Andrews et al.	6,041,381	A	3/2000	Hoese
5,634,127	A	5/1997	Cloud et al.	6,044,438	A	3/2000	Olnowich
5,642,482	A	6/1997	Pardillos	6,047,323	A	4/2000	Krause
5,664,114	A	9/1997	Krech, Jr. et al.	6,047,356	A	4/2000	Anderson et al.
5,671,355	A	9/1997	Collins	6,049,528	A	4/2000	Hendel et al.
5,678,060	A	10/1997	Yokoyama et al.	6,057,863	A	5/2000	Olari
5,682,534	A	10/1997	Kapoor et al.	6,061,368	A	5/2000	Hitzelberger
5,684,954	A	11/1997	Kaiserwerth et al.	6,065,096	A	5/2000	Day et al.
5,692,130	A	11/1997	Shobu et al.	6,067,569	A	5/2000	Khaki et al.
5,699,317	A	12/1997	Sartore et al.	6,070,200	A	5/2000	Gates et al.
5,699,350	A	12/1997	Kraslavsky	6,078,564	A	6/2000	Lakshman et al.
5,701,434	A	12/1997	Nakagawa	6,078,733	A	6/2000	Osborne
5,701,516	A	12/1997	Cheng et al.	6,097,734	A	8/2000	Gotesman et al.
5,706,514	A	1/1998	Bonola	6,101,555	A	8/2000	Goshey et al.
5,727,142	A	3/1998	Chen	6,111,673	A	8/2000	Chang et al.
5,742,765	A	4/1998	Wong et al.	6,115,615	A	9/2000	Ota et al.
5,749,095	A	5/1998	Hagersten	6,122,670	A	9/2000	Bennett et al.
5,751,715	A	5/1998	Chan et al.	6,141,701	A	10/2000	Whitney
5,751,723	A	5/1998	Vanden Heuvel et al.	6,141,705	A	10/2000	Anand et al.
5,752,078	A	5/1998	Delp et al.	6,145,017	A	11/2000	Ghaffari
5,758,084	A	5/1998	Silverstein et al.	6,157,944	A	12/2000	Pedersen
5,758,089	A	5/1998	Gentry et al.	6,157,955	A	12/2000	Narad et al.
5,758,186	A	5/1998	Hamilton et al.	6,172,980	B1	1/2001	Flanders et al.
5,758,194	A	5/1998	Kuzma	6,173,333	B1	1/2001	Jolitz et al.
5,768,618	A	6/1998	Erickson et al.	6,181,705	B1	1/2001	Branstad et al.
5,771,349	A	6/1998	Picazo, Jr. et al.	6,202,105	B1	3/2001	Gates et al.
5,774,660	A	6/1998	Brendel et al.	6,219,693	B1	4/2001	Napolitano et al.
5,778,013	A	7/1998	Jedwab	6,223,242	B1	4/2001	Sheafor et al.
5,778,419	A	7/1998	Hansen et al.	6,226,680	B1	5/2001	Boucher et al.
5,790,804	A	8/1998	Osborne	6,233,242	B1	5/2001	Mayer et al.
5,794,061	A	8/1998	Hansen et al.	6,243,667	B1	6/2001	Kerr et al.
5,799,150	A	8/1998	Hamilton et al.	6,246,683	B1	6/2001	Connery et al.
5,802,258	A	9/1998	Chen	6,247,060	B1	6/2001	Boucher et al.
5,802,580	A	9/1998	McAlpine	6,279,051	B1	8/2001	Gates et al.
5,809,328	A	9/1998	Nogales et al.	6,289,023	B1	9/2001	Dowling et al.
5,809,527	A	9/1998	Cooper et al.	6,298,403	B1	10/2001	Suri et al.
5,812,775	A	9/1998	Van Seters et al.	6,324,649	B1	11/2001	Eyres et al.
5,815,646	A	9/1998	Purcell et al.	6,334,153	B2	12/2001	Boucher et al.
5,819,111	A	10/1998	Davies et al.	6,343,345	B1	1/2002	Hilla et al.
5,828,835	A	10/1998	Isfeld et al.	6,343,360	B1	1/2002	Feinleib
5,848,293	A	12/1998	Gentry et al.	6,345,301	B1	2/2002	Burns et al.
5,870,394	A	2/1999	Oprea	6,345,302	B1	2/2002	Bennett et al.
5,872,919	A	2/1999	Wakeland et al.	6,356,951	B1	3/2002	Gentry et al.
5,878,225	A	3/1999	Bilansky et al.	6,370,599	B1	4/2002	Anand et al.
5,878,227	A	3/1999	Wade et al.	6,385,647	B1	5/2002	Willis et al.
5,892,903	A	4/1999	Klaus	6,389,468	B1	5/2002	Muller et al.
5,898,713	A	4/1999	Melzer et al.	6,389,479	B1	5/2002	Boucher et al.
5,913,028	A	6/1999	Wang et al.	6,393,487	B2	5/2002	Boucher et al.
5,915,094	A	6/1999	Kouloheris et al.	6,418,169	B1	7/2002	Datari
5,917,828	A	6/1999	Thompson	6,421,742	B1	7/2002	Tillier
5,920,566	A	7/1999	Hendel et al.	6,421,753	B1	7/2002	Hoese et al.

US 8,805,948 B2

Page 3

6,427,169	B1	7/2002	Elzur	7,627,001	B2	12/2009	Craft et al.
6,427,171	B1	7/2002	Craft et al.	7,627,684	B2	12/2009	Boucher et al.
6,427,173	B1	7/2002	Boucher et al.	7,640,364	B2	12/2009	Craft et al.
6,434,620	B1	8/2002	Boucher et al.	7,664,868	B2	2/2010	Boucher et al.
6,434,651	B1	8/2002	Gentry, Jr.	7,664,883	B2	2/2010	Craft et al.
6,449,656	B1	9/2002	Elzur et al.	7,673,072	B2	3/2010	Boucher et al.
6,452,915	B1	9/2002	Jorgensen	7,694,024	B2	4/2010	Philbrick et al.
6,453,360	B1	9/2002	Muller et al.	7,738,200	B2	6/2010	Jones et al.
6,453,406	B1	9/2002	Sarnikowski et al.	2001/0004354	A1	6/2001	Jolitz
6,470,415	B1	10/2002	Starr et al.	2001/0013059	A1	8/2001	Dawson et al.
6,473,425	B1	10/2002	Bellaton et al.	2001/0014892	A1	8/2001	Gaither et al.
6,480,489	B1	11/2002	Muller et al.	2001/0014954	A1	8/2001	Purcell et al.
6,483,804	B1	11/2002	Muller et al.	2001/0025315	A1	9/2001	Jolitz
6,487,202	B1	11/2002	Klausmeier et al.	2001/0037406	A1	11/2001	Philbrick et al.
6,487,654	B2	11/2002	Dowling	2001/0048681	A1	12/2001	Bilic et al.
6,490,631	B1	12/2002	Teich et al.	2001/0053148	A1	12/2001	Bilic et al.
6,502,144	B1	12/2002	Accarie	2002/0073223	A1	6/2002	Darnell et al.
6,523,119	B2	2/2003	Pavlin et al.	2002/0112175	A1	8/2002	Makofka et al.
6,526,446	B1	2/2003	Yang et al.	2002/0156927	A1	10/2002	Boucher et al.
6,542,504	B1	4/2003	Mahler et al.	2003/0014544	A1	1/2003	Petty
6,570,884	B1	5/2003	Connery et al.	2003/0046330	A1	3/2003	Hayes
6,591,302	B2	7/2003	Boucher et al.	2003/0066011	A1	4/2003	Oren
6,591,310	B1	7/2003	Johnson	2003/0067903	A1	4/2003	Jorgensen
6,594,261	B1	7/2003	Boura et al.	2003/0110344	A1	6/2003	Szezepanek et al.
6,631,484	B1	10/2003	Born	2003/0165160	A1	9/2003	Minami et al.
6,648,611	B2	11/2003	Morse et al.	2004/0010712	A1	1/2004	Hui et al.
6,650,640	B1	11/2003	Muller et al.	2004/0042458	A1	3/2004	Elzu
6,657,757	B1	12/2003	Chang et al.	2004/0042464	A1	3/2004	Elzur et al.
6,658,480	B2	12/2003	Boucher et al.	2004/0049580	A1	3/2004	Boyd et al.
6,678,283	B1	1/2004	Teplitsky	2004/0049601	A1	3/2004	Boyd et al.
6,681,364	B1	1/2004	Calvignac et al.	2004/0054814	A1	3/2004	McDaniel
6,683,851	B1	1/2004	Willkie et al.	2004/0059926	A1	3/2004	Angelo et al.
6,687,758	B2	2/2004	Craft et al.	2004/0088262	A1	5/2004	Boucher et al.
6,697,366	B1	2/2004	Kim	2004/0153578	A1	8/2004	Elzur
6,697,868	B2	2/2004	Craft et al.	2004/0210795	A1	10/2004	Anderson
6,751,665	B2	6/2004	Philbrick et al.	2004/0213290	A1	10/2004	Johnson et al.
6,757,746	B2	6/2004	Boucher et al.	2004/0246974	A1	12/2004	Gyugyi et al.
6,765,901	B1	7/2004	Johnson et al.	2004/0249957	A1	12/2004	Egis et al.
6,807,581	B1	10/2004	Starr et al.	2005/0060538	A1	3/2005	Beverly
6,842,896	B1	1/2005	Redding et al.	2005/0144300	A1	6/2005	Craft et al.
6,862,264	B1	3/2005	Moura et al.	2006/0133386	A1	6/2006	McCormack et al.
6,912,522	B2	6/2005	Edgar	2006/0248208	A1	11/2006	Walbeck et al.
6,938,092	B2	8/2005	Burns	2007/0083682	A1	4/2007	Bartley et al.
6,941,386	B2	9/2005	Craft et al.	2007/0140240	A1	6/2007	Dally et al.
6,965,941	B2	11/2005	Boucher et al.	2008/0043732	A1	2/2008	Desai et al.
6,976,148	B2	12/2005	Arimilli et al.	2008/0170501	A1	7/2008	Patel et al.
6,996,070	B2	2/2006	Starr et al.	2008/0209084	A1	8/2008	Wang et al.
7,016,361	B2	3/2006	Swonk et al.	2008/0240111	A1	10/2008	Gadelrab
7,042,898	B2	5/2006	Blightman et al.	2009/0063696	A1	3/2009	Wang et al.
7,047,320	B2	5/2006	Arimilli et al.				
7,073,196	B1	7/2006	Dowd et al.				
7,076,568	B2	7/2006	Philbrick et al.				
7,089,326	B2	8/2006	Boucher et al.				
7,093,099	B2	8/2006	Bodas et al.				
7,124,205	B2	10/2006	Craft et al.				
7,133,940	B2	11/2006	Blightman et al.				
7,167,926	B1	1/2007	Boucher et al.				
7,167,927	B2	1/2007	Philbrick et al.				
7,174,393	B2	2/2007	Boucher et al.				
7,181,531	B2	2/2007	Pinkerton et al.				
7,185,266	B2	2/2007	Blightman et al.				
7,187,679	B2	3/2007	Dally et al.				
7,191,241	B2	3/2007	Boucher et al.				
7,191,318	B2	3/2007	Tripathy et al.				
7,237,036	B2	6/2007	Boucher et al.				
7,254,696	B2	8/2007	Mittal et al.				
7,260,518	B2	8/2007	Kerr et al.				
7,283,522	B2	10/2007	Siddabathuni				
7,284,070	B2	10/2007	Boucher et al.				
7,287,092	B2	10/2007	Sharp				
7,337,241	B2	2/2008	Boucher et al.				
7,461,160	B2	12/2008	Boucher et al.				
7,472,156	B2	12/2008	Philbrick et al.				
7,496,689	B2	2/2009	Sharp et al.				
7,502,869	B2	3/2009	Boucher et al.				
7,519,699	B2	4/2009	Jain				
7,543,087	B2	6/2009	Philbrick et al.				
7,584,260	B2	9/2009	Craft et al.				
7,620,726	B2	11/2009	Craft et al.				

FOREIGN PATENT DOCUMENTS

EP	574140	A1	12/1993
WO	WO 98/19412		5/1998
WO	WO 98/50852		11/1998
WO	WO 99/04343		1/1999
WO	WO 99/65219		12/1999
WO	PCT/US98/24943		3/2000
WO	WO 00/13091		3/2000
WO	WO 01/04770		1/2001
WO	WO 01/05107		1/2001
WO	WO 01/05116		1/2001
WO	WO 01/05123		1/2001
WO	WO 01/40960		6/2001
WO	WO 2007/130476		11/2007

OTHER PUBLICATIONS

U.S. Appl. No. 60/053,240, filed Jul. 18, 1997, Jolitz et al.
Form 10-K for Exelan, Inc., for the fiscal year ending Dec. 31, 1987 (10 pages).
Form 10-K for Exelan, Inc., for the fiscal year ending Dec. 31, 1988 (10 pages).
Internet pages entitled: Technical White Paper—Xpoint's Disk to LAN Acceleration Solution for Windows NT Server, printed Jun. 5, 1997.
Jato Technologies Internet pages entitled: Network Accelerator Chip Architecture, twelve-slide presentation, printed Aug. 19, 1998.

- EETIMES article entitled: Enterprise System Uses Flexible Spec, dated Aug. 10, 1998, printed Nov. 25, 1998.
- Internet pages entitled iReady About Us and iReady Products, printed Nov. 25, 1998.
- Internet pages entitled: Smart Ethernet Network Interface Cards which Berend Ozceri is developing, and Internet pages entitled: Hardware Assisted Protocol Processing, which Eugene Feinberg is working on, printed Nov. 25, 1998.
- Internet pages of Xaqti corporation entitled: GigaPower Protocol Processor Product review, printed Nov. 25, 2009.
- Internet pages entitled: DART: Fast Application Level Networking via Data-copy Avoidance, by Robert J. Walsh, printed Jun. 3, 1999.
- Internet pages of InterProphet entitled: Frequently Asked Questions, by Lynne Jolitz, printed Jun. 14, 2000.
- Andrew S. Tanenbaum, Computer Networks, Third Edition, 1996, ISBN: 0-13-349945-6.
- iReady Corporation, article entitled "The 1-1000 Internet Tuner", 2 pages, date unknown.
- iReady News Archive article entitled "Seiko Instruments Inc. (SII) Introduces World's 2 First Internet-Ready Intelligent LCD Modules Based on iReady Technology," Santa Clara, CA and Chiba, Japan, Oct. 26, 1998. 2 pages, printed Nov. 2, 1998.
- WindRiver article entitled "Tornado: For Intelligent Network Acceleration", copyright Wind River Systems, 2001, 2 pages.
- WindRiver White Paper entitled "Complete TCP/IP Offload for High-Speed Ethernet Networks", Copyright Wind River Systems, 2002, 7 pages.
- Intel article entitled "Solving Server Bottlenecks with Intel Server Adapters", Copyright Intel Corporation, 1999, 8 pages.
- Chandranmenon, Girish P. et al. "Trading Packet Headers for Packet Processing." IEEE/ACM Transactions on Networking. vol. 4, No. 2. Apr. 1996. pp. 141-152.
- Starr, David D. et al. "Intelligent Network Storage Interface Device." U.S. Appl. No. 09/675,700, filed Sep. 29, 2000.
- Boucher, Laurence B. et al. "Intelligent Network Interface System and Method for Accelerated Protocol Processing." U.S. Appl. No. 09/692,561, filed Oct. 18, 2000.
- Starr, Daryl D., et al. "Intelligent Network Storage Interface System." U.S. Appl. No. 10/261,051, filed Sep. 30, 2002.
- Merritt, Rick. "Ethernet Interconnect Outpacing Infiniband at Intel." EE Times article. Sep. 11, 2002. 9 pages.
- Craft, Peter K. et al. "TCP Offload Device that Batches Session Layer Headers to Reduce Interrupts as Well as CPU Copies." U.S. Appl. No. 12/581,342, filed Oct. 19, 2009.
- Craft, Peter K. et al. "TCP Offload Send Optimization." U.S. Appl. No. 12/504,021, filed Jul. 16, 2009.
- Philbrick, Clive M. et al. "Freeing Transmit Memory on a Network Interface Device Prior to Receiving an Acknowledgment That Transmit Data Has Been Received by a Remote Device." U.S. Appl. No. 12/470,980, filed May 22, 2009.
- Starr, Daryl D. et al. "Accelerating Data Transfer in a Virtual Computer System with Tightly Coupled TCP Connections." U.S. Appl. No. 12/410,366, filed Mar. 24, 2009.
- Boucher, Laurence B. et al. "Obtaining a Destination Address So That a Network Interface Device Can Write Network Data Without Headers Directly Into Host Memory." U.S. Appl. No. 12/325,941, filed Dec. 1, 2008.
- Boucher, Laurence B. et al. "Enabling an Enhanced Function of an Electronic Device." U.S. Appl. No. 11/985,948, filed Nov. 19, 2007.
- Starr, Daryl D. et al. "Network Interface Device With 10 Gb/s Full-Duplex Transfer Rate." U.S. Appl. No. 11/799,720, filed May 1, 2007.
- Craft, Peter K. et al. "Peripheral Device That DMAS the Same Data to Different Locations in a Computer." U.S. Appl. No. 11/788,719, filed Apr. 19, 2007.
- Boucher, Laurence B. et al. "TCP/IP Offload Network Interface Device." U.S. Appl. No. 11/701,705, filed Feb. 2, 2007.
- Starr, Daryl D. et al. "TCP/IP Offload Device With Reduced Sequential Processing." U.S. Appl. No. 11/348,810, filed Feb. 6, 2006.
- Craft, Peter K. et al. "Transferring Control of TCP Connections Between Hierarchy of Processing Mechanisms." U.S. Appl. No. 11/249,006, filed Oct. 11, 2005.
- Boucher, Laurence B. et al. "Network Interface Device That Can Transfer Control of a TCP Connection to a Host CPU." U.S. Appl. No. 11/029,863, filed Jan. 4, 2005.
- Craft, Peter K. et al. "Protocol Stack That Offloads a TCP Connection From a Host Computer to a Network Interface Device." U.S. Appl. No. filed Dec. 30, 2004.
- Craft, Peter K. et al. "Protocol Stack That Offloads a TCP Connection From a Host Computer to a Network Interface Device." U.S. Appl. No. 11/016,642, filed Dec. 16, 2004.
- Boucher, Laurence B. et al. "Method and Apparatus for Dynamic Packet Batching With a High Performance Network Interface." U.S. Appl. No. 0/678,336, filed Oct. 3, 2003.
- Philbrick Clive M. et al. "Method and Apparatus for Data Re-Assembly with a High Performance Network Interface." U.S. Appl. No. 10/634,062, filed Aug. 4, 2003.
- Boucher, Laurence B. et al. "High Network Interface Device System for Accelerated Communication." U.S. Appl. No. 10/601,237, filed Jun. 19, 2003.
- Boucher, Laurence B. et al. "Method and Apparatus for Distributing Network Traffic Processing on a Multiprocessor Computer." U.S. Appl. No. 10/438,719, filed May 14, 2003.
- Boucher, Laurence B. et al. "Parsing a Packet Header." U.S. Appl. No. 10/277,604, filed Oct. 18, 2002.
- Internet pages entitled "Hardware Assisted Protocol Processing", (which Eugene Feinberg is working on), 1 page, printed Nov. 25, 1998.
- Zilog product Brief entitled "Z85C30 CMOS SCC Serial Communication Controller", Zilog Inc., 3 pages, 1997.
- Internet pages of Xpoint Technologies, Inc. entitled "Smart LAN Work Requests", 5 pages, printed Dec. 19, 1997.
- Internet pages entitled: Asante and 100BASE-T Fast Ethernet. 7 Pages, printed May 27, 1997.
- Internet pages entitled: A Guide to the Paragon XP/S-A7 Supercomputer at Indiana University. 13 pages printed Dec. 21, 1998.
- Richard Stevens, "TCP/IP Illustrated, vol. 1, The Protocols", pp. 325-326 (1994).
- Internet pages entitled: Northridge/Southbridge vs. Intel Hub Architecture, 4 pages, printed Feb. 19, 2001.
- Gigabit Ethernet Technical Brief, Achieving End-to-End Performance. Alteon Networks, Inc., First Edition, Sep. 1996.
- Internet pages directed to Technical Brief on Alteon Ethernet Gigabit NIC technology, www.alteon.com, 14 pages, printed Mar. 15, 1997.
- VIA Technologies, Inc. article entitled "VT8501 Apollo MVP4", pp. i-iv, 1-11, cover and copyright page, revision 1.3, Feb. 1, 2000.
- iReady News Archives article entitled "iReady Rounding Out Management Team with Two Key Executives", <http://www.ireadyco.com/archives/keyexec.html>, 2 pages, printed Nov. 28, 1998.
- "Toshiba Delivers First Chips to Make Consumer Devices Internet-Ready Based on iReady's Design," Press Release Oct. 1998, 3 pages, printed Nov. 28, 1998.
- Internet pages from iReady Products, web site: <http://www.ireadyco.com/products.html>, 2 pages, downloaded Nov. 25, 1998.
- iReady News Archives, Toshiba, iReady shipping Internet chip, 1 page, printed Nov. 25, 1998.
- Interprophet article entitled "Technology", <http://www.interprophet.com/technology.html>, 17 pages, printed Mar. 1, 2000.
- iReady Corporation, article entitled "The 1-1000 Internet Tuner", 2 pages, date unknown.
- iReady article entitled "About Us Introduction", Internet pages from <http://www.ireadyco.com/about.html>, 3 pages, printed Nov. 25, 1998.
- iReady News Archive article entitled "Revolutionary Approach to Consumer Electronics Internet Connectivity Funded", San Jose, CA, Nov. 20, 1997. 2 pages, printed Nov. 2, 1998.
- iReady News Archive article entitled "Seiko Instruments Inc. (SII) Introduces World's First Internet-Ready Intelligent LCD Modules Based on iReady Technology," Santa Clara, CA and Chiba, Japan, Oct. 26, 1998. 2 pages, printed Nov. 2, 1998.
- NEWSwatch article entitled "iReady internet Tuner to Web Enable Devices", Tuesday, Nov. 5, 1996, printed Nov. 2, 1998, 2 pages.
- EETimes article entitled "Tuner for Toshiba, Toshiba Taps iReady for Internet Tuner", by David Lammers, 2 pages, printed Nov. 2, 1998.

- "Comparison of Novell Netware and TCP/IP Protocol Architectures", by J.S. Carbone, 19 pages, printed Apr. 10, 1998.
- Adaptec article entitled "AEA-7110C-A DuraSAN product", 11 pages, printed Oct. 1, 2001.
- iSCSI HBA article entitled "iSCSI and 2Gigabit fibre Channel Host Bus Adapters from Emulex, QLogic, Adaptec, JNT", 8 pages, printed Oct. 1, 2001.
- iSCSI HBA article entitled "FCE-3210/6410 32 and 64-bit PCI-to-Fibre Channel HBA", 6 pages, printed Oct. 1, 2001.
- iSCSI HBA article entitled "iSCSI Storage", 6 pages, printed Oct. 1, 2001.
- "Two-Way TCP Traffic Over Rate Controlled Channels: Effects and Analysis", by Kalampoukas et al., IEEE Transactions on Networking, vol. 6, No. 6, Dec. 1998.
- iReady News article entitled "Toshiba Delivers First Chips to Make Consumer Devices Internet-Ready Based on iReady Design", Santa Clara, CA and Tokyo, Japan, Oct. 14, 1998, printed Nov. 2, 1998.
- U.S. Appl. No. 08/964,304, by Napolitano, et al., entitled "File Array Storage Architecture", filed Nov. 4, 1997.
- "File System Design for an NFS File Server Appliance", Article by D. Hitz, et al., 13 pages.
- Adaptec Press Release article entitled "Adaptec Announces EtherStorage Technology", 2 pages, May 4, 2000, printed Jun. 14, 2000.
- Adaptec article entitled "EtherStorage Frequently Asked Questions", 5 pages, printed Jul. 19, 2000.
- Adaptec article entitled "EtherStorage White Paper", 7 pages, printed Jul. 19, 2000.
- CIBC World Markets article entitled "Computers; Storage", by J. Bertino et al., 9 pages, dated Aug. 7, 2000.
- Merrill Lynch article entitled "Storage Futures", by S. Milunovich, 22 pages, dated May 10, 2000.
- CBS Market Watch article entitled "Montreal Start-Up Battles Data Storage Bottleneck", by S. Taylor, dated Mar. 5, 2000, 2 pages, printed Mar. 7, 2000.
- Internet-draft article entitled "SCSI/TCP (SCSI over TCP)", by J. Satran et al., 38 pages, dated Feb. 2000, printed May 19, 2000.
- Article entitled LRP: A New Network Subsystem Architecture for Server Systems, by P. Druschel and G. Banga, Rice University, 15 pages.
- Internet RFC/STD/FYI/BCP Archives article with heading "RFC2140" entitled "TCP control Block Interdependence", web address <http://www.faqs.org/rfcs/rfc2140.html>, 9 pages, printed Sep. 20, 2002.
- I. Heizer et al., Common Internet File System Protocol (CIFS/1.0), Internet Draft, Internet Engineering Task Force (IETF), Jun. 13, 1996.
- David D. Clark et al., NETBLT: A Bulk Data Transfer Protocol, Request for Comments (RFC) 998, Mar. 1987.
- K. Sollins, The TFTP Protocol (Revision 2), Request for Comments (RFC) 1350, Jul. 1992.
- W. David Schwaderer, C Programmer's Guide to NetBIOS, IPX, and SPX, 1992, pp. 322-323.
- Internet pages entitled Technical White Paper-Xpoint's Disk to LAN Acceleration Solution for Windows NT Server, printed Jun. 5, 1997, 15 pages.
- Jato Technologies article entitled Network Accelerator Chip Architecture, twelve-slide presentation, printed Aug. 19, 1998, 13 pages.
- EETimes article entitled Enterprise System Uses Flexible Spec, dated Aug. 10, 1998, printed Nov. 25, 1998, 3 pages.
- Internet pages entitled "Smart Ethernet Network Interface Cards", which Berend Ozceri is developing, printed Nov. 25, 1998, 2 pages.
- Internet pages of Xahti corporation entitled "GigaPower Protocol Processor Product Review", printed Nov. 25, 1999, 4 pages.
- U.S. Appl. No. 60/283,896, Titled: CRC Calculations for Out of Order PUDs, filed Apr. 12, 2003, Inventor: Amit Oren, Assignee: Siliquent Technologies Ltd.
- Internet pages entitled "DART: Fast Application Level Networking via Data-Copy Avoidance," by Robert J. Walsh, printed Jun. 3, 1999, 25 pages.
- Andrew S. Tanenbaum, Computer Networks, Third Edition, 1996, ISBN 0-13-349945-6.
- Article from Rice University entitled "LRP: A New Network Subsystem Architecture for Server Systems", by Peter Druschel and Gaurav Banga, 14 pages.
- Internet RFD/STD/FYI/BCP Archives article with heading "RFC2140" entitled "TCP Control Block Interdependence", web address <http://www.faqs.org/rfcs/rfc2140.html>, 9 pages, printed Sep. 20, 2002.
- Wind River article entitled "Tornado: For Intelligent Network Acceleration", copyright Wind River Systems, 2001, 2 pages.
- Schwaderer et al., IEEE Computer Society Press publication entitled, "XTP in VLSI Protocol Decomposition for ASIC Implementation", from 15th Conference on Local Computer Networks, 5 pages, Sep. 30-Oct. 3, 1990.
- Beach, Bob, IEEE Computer Society Press publication entitled, "UltraNet: An Architecture for Gigabit Networking", from 15th Conference on Local Computer Networks, 18 pages, Sep. 30-Oct. 3, 1990.
- Chesson et al., IEEE Symposium Record entitled, "The Protocol Engine Chipset", from Hot Chips III, 16 pages, Aug. 26-27, 1991.
- Maclean et al., IEEE Global Telecommunications Conference, Globecom '91, presentation entitled, "An Outboard Processor for High Performance Implementation of Transport Layer Protocols", 7 pages, Dec. 2-5, 1991.
- Ross et al., IEEE article entitled "FX1000: A high performance single chip Gigabit Ethernet NIC", from Compean '97 Proceedings, 7 pages, Feb. 23-26, 1997.
- Strayer et al., "Ch. 9: The Protocol Engine" from XTP: The Transfer Protocol, 12 pages, Jul. 1992.
- Publication entitled "Protocol Engine Handbook", 44 pages, Oct. 1990.
- Koufopavlou et al., IEEE Global Telecommunications Conference, Globecom '92, presentation entitled, "Parallel TCP for High Performance Communication Subsystems", 7 pages, Dec. 6-9, 1992.
- Lilienkamp et al., Publication entitled "Proposed Host-Front End Protocol", 56 pages, Dec. 1984.
- Thia et al. Publication entitled "High-Speed OSI Protocol Bypass Algorithm with Window Flow Control," Protocols for High Speed Networks, pp. 53-68, 1993.
- U.S. Appl. No. 60/053,240, Titled: TCP/IP Network Accelerator and Method of Use, filed Jul. 17, 1997, Inventor: William Jolitz et al.
- Thia et al. Publication entitled "A Reduced Operational Protocol Engine (ROPE) for a multiple-layer bypass architecture," Protocols for High Speed Networks, pp. 224-239, 1995.
- Thia, Y.H. Publication entitled "High-Speed OSI Protocol Bypass Algorithm with Window Flow Control", Protocols for High Speed Networks, pp. 53-68, 1993.
- Thia, Y.H. Publication entitled "A Reduced Operational Protocol Engine (ROPE) for a multiple layer bypass architecture", Protocols for High Speed Networks, pp. 224-239, 1995.

Primary Examiner — Moustafa M Meky

(74) Attorney, Agent, or Firm — Mark Lauer; Silicon Edge Law Group LLP

(57)

ABSTRACT

A system for protocol processing in a computer network has an intelligent network interface card (INIC) or communication processing device (CPD) associated with a host computer. The INIC or CPD provides a fast-path that avoids host protocol processing for most large multipacket messages, greatly accelerating data communication. The INIC or CPD also assists the host for those message packets that are chosen for processing by host software layers. A communication control block (CCB) for a message is defined that allows DMA controllers of the INIC to move data, free of headers, directly to or from a destination or source in the host. The CCB can be passed back to the host for message processing by the host. The INIC or CPD contains hardware circuits configured for protocol processing that can perform that specific task faster than the host CPU. One embodiment includes a processor providing transmit, receive and management processing, with full duplex communication for four fast Ethernet nodes.

22 Claims, 14 Drawing Sheets

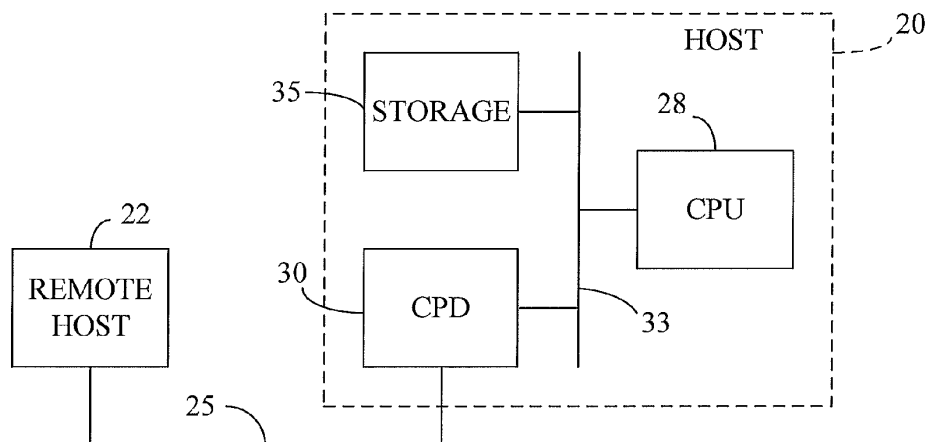


FIG. 1

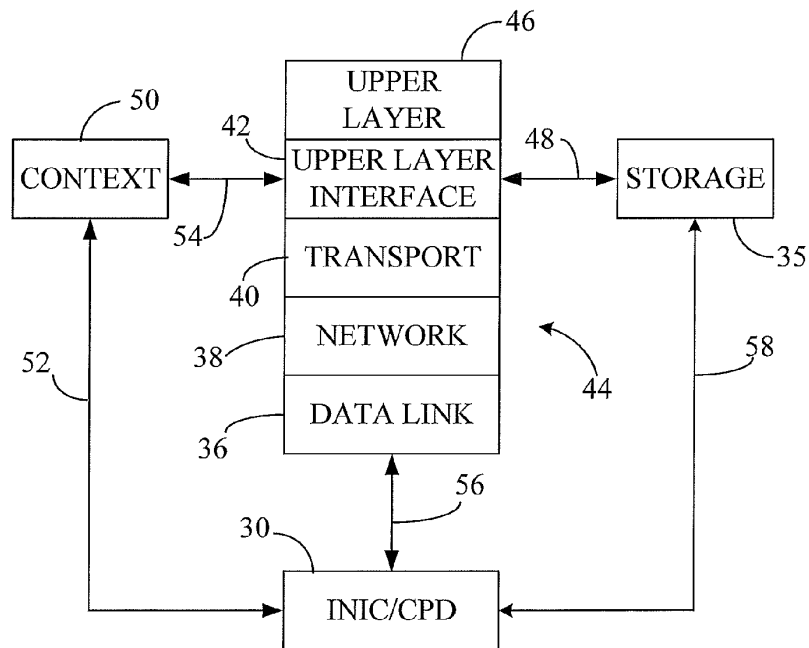


FIG. 2

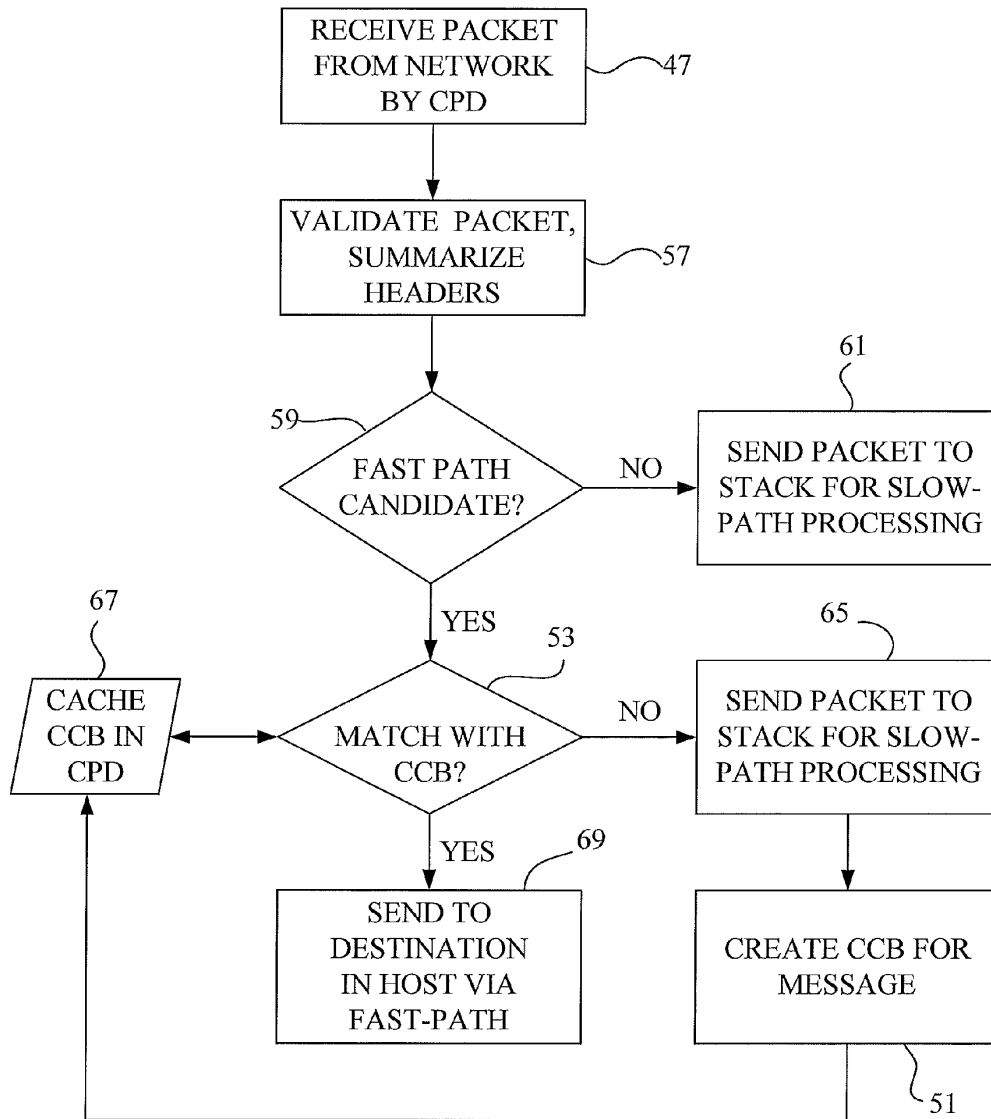


FIG. 3

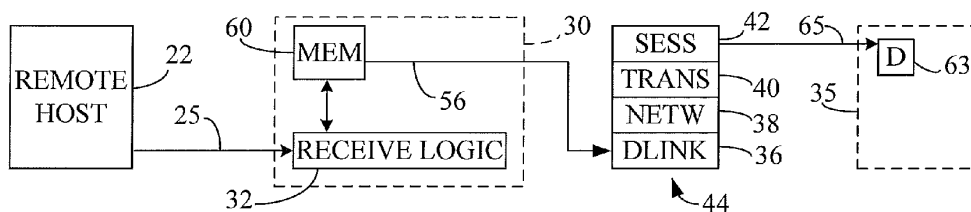


FIG. 4A

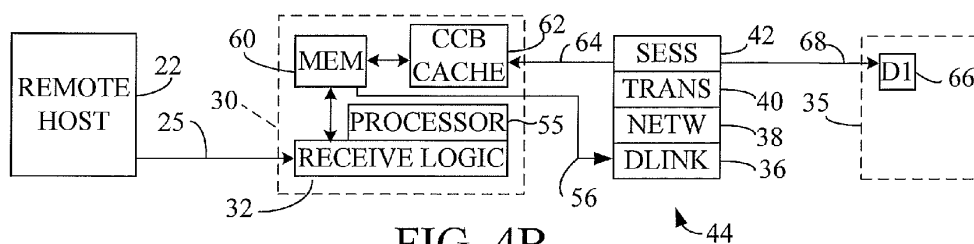


FIG. 4B

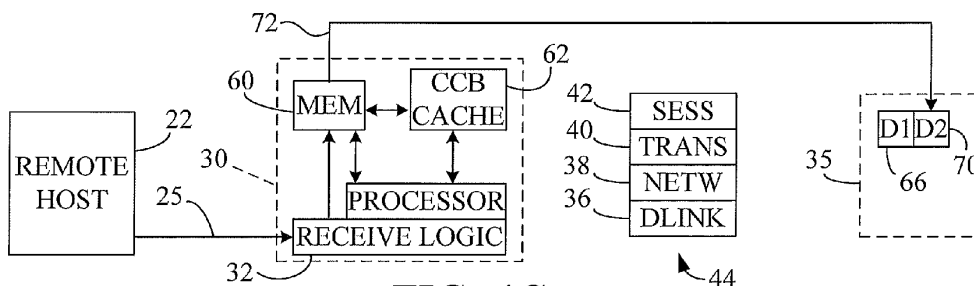


FIG. 4C

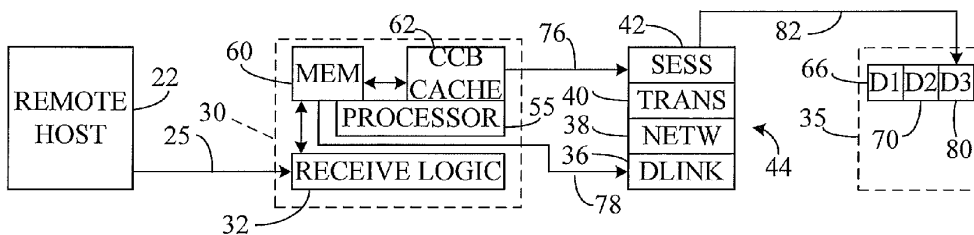


FIG. 4D

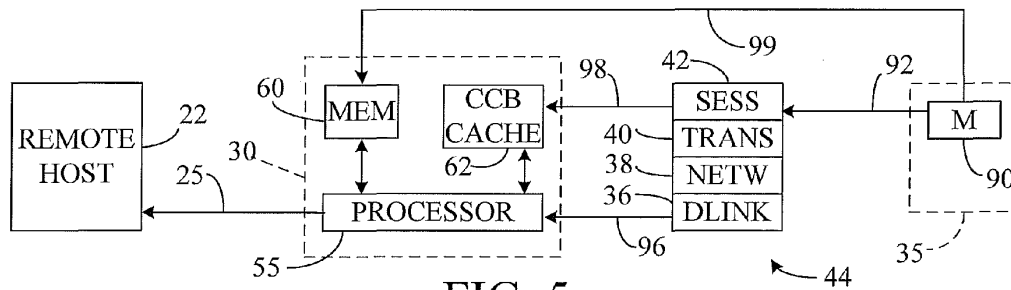


FIG. 5

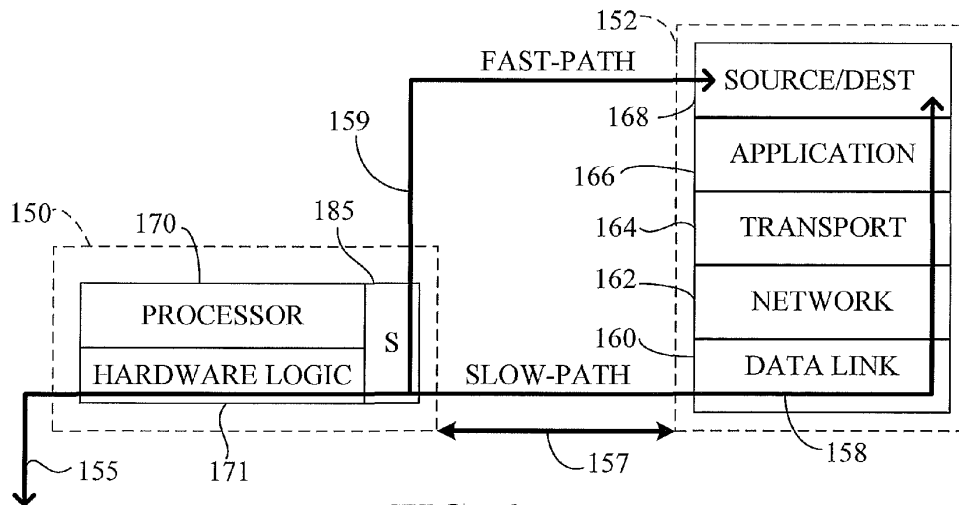


FIG. 6

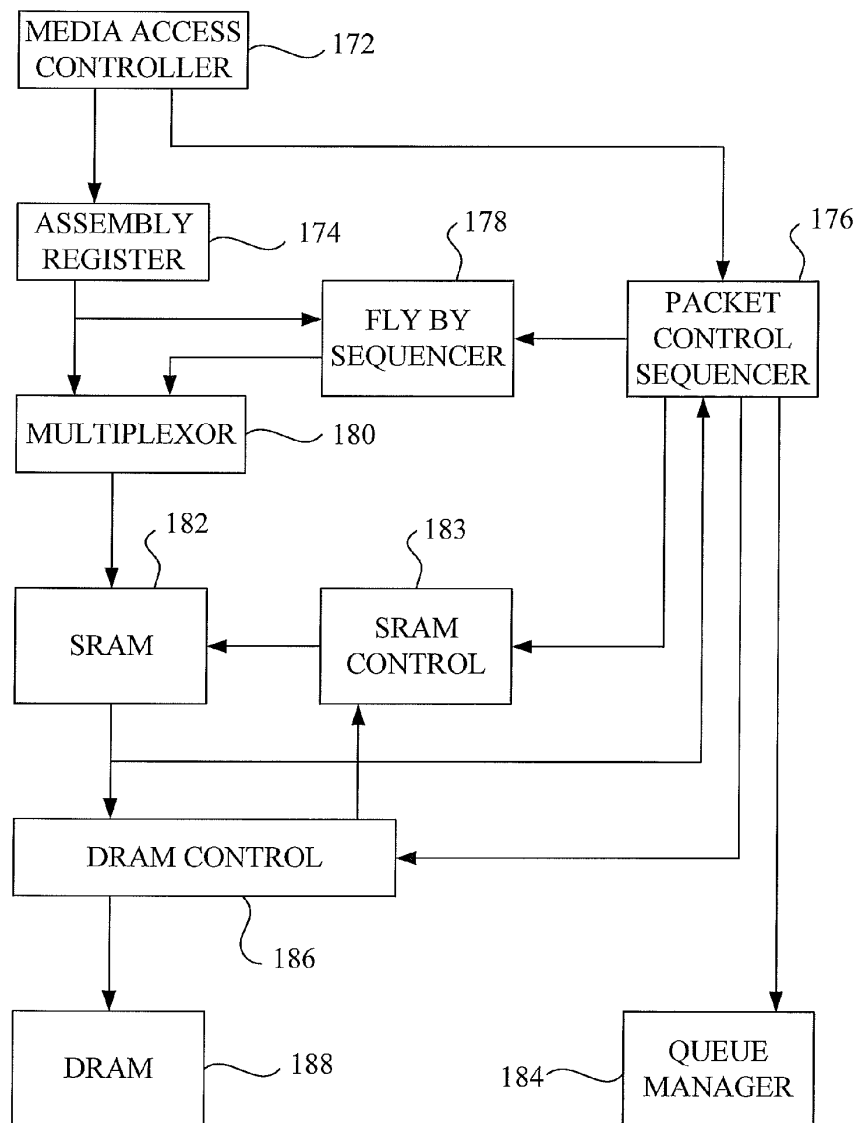


FIG. 7

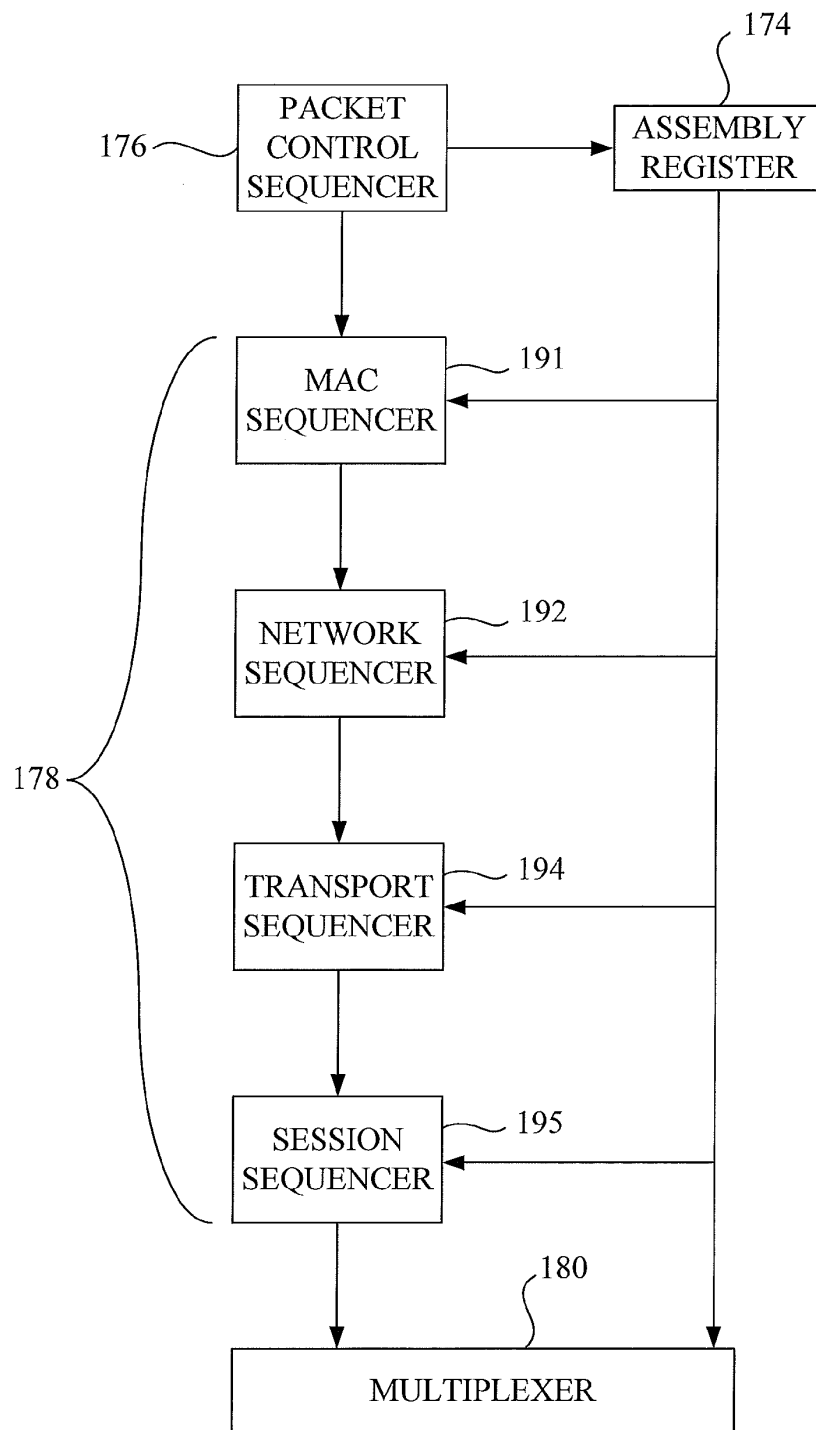


FIG. 8

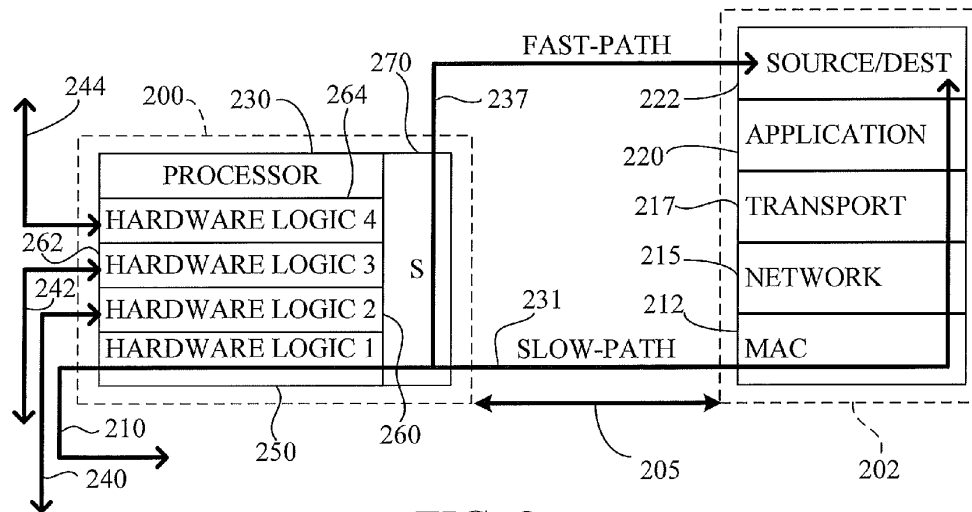


FIG. 9

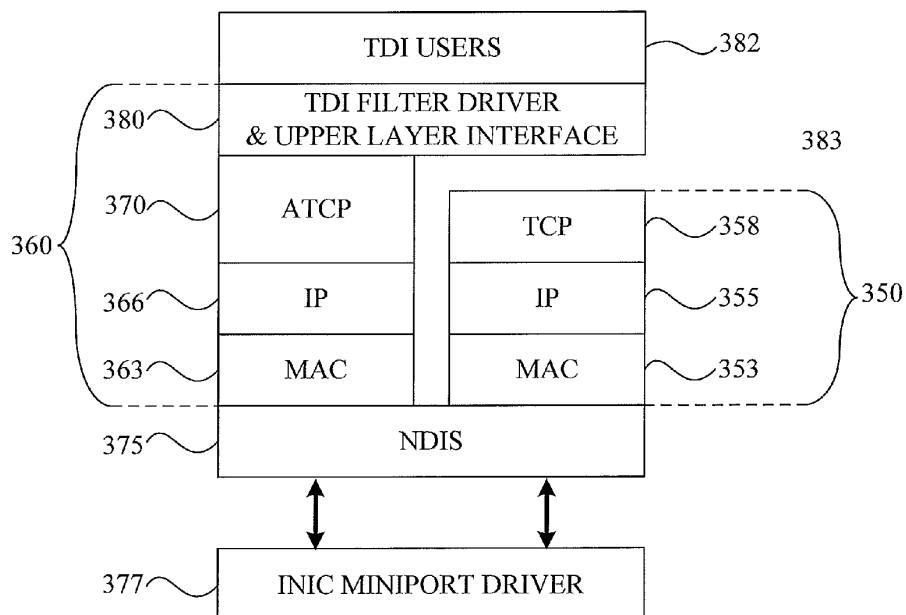


FIG. 11

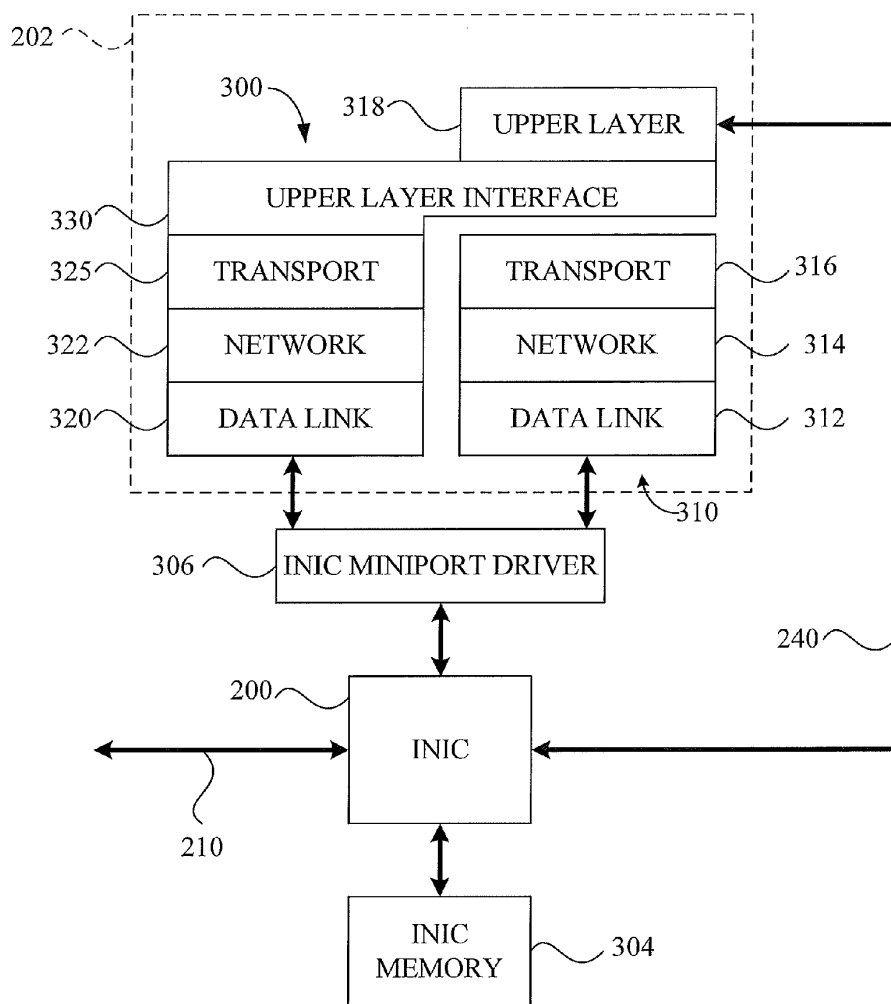


FIG. 10

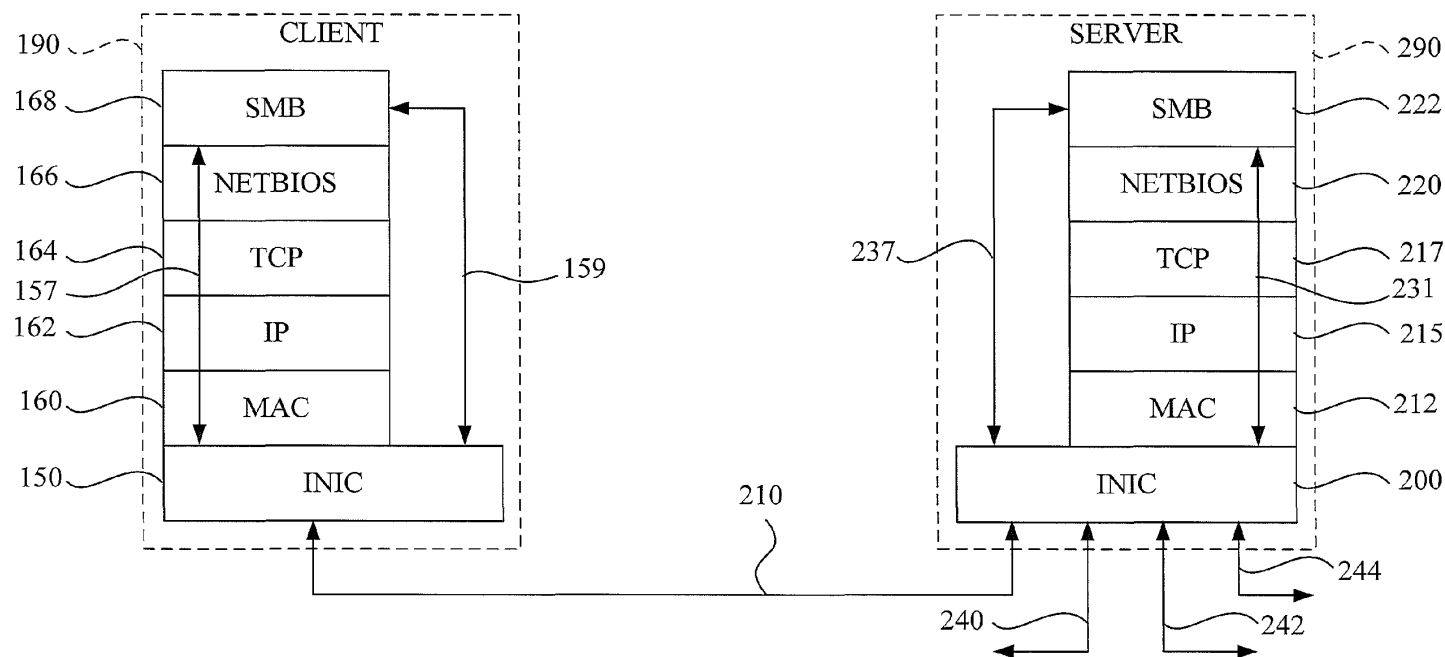


FIG. 12

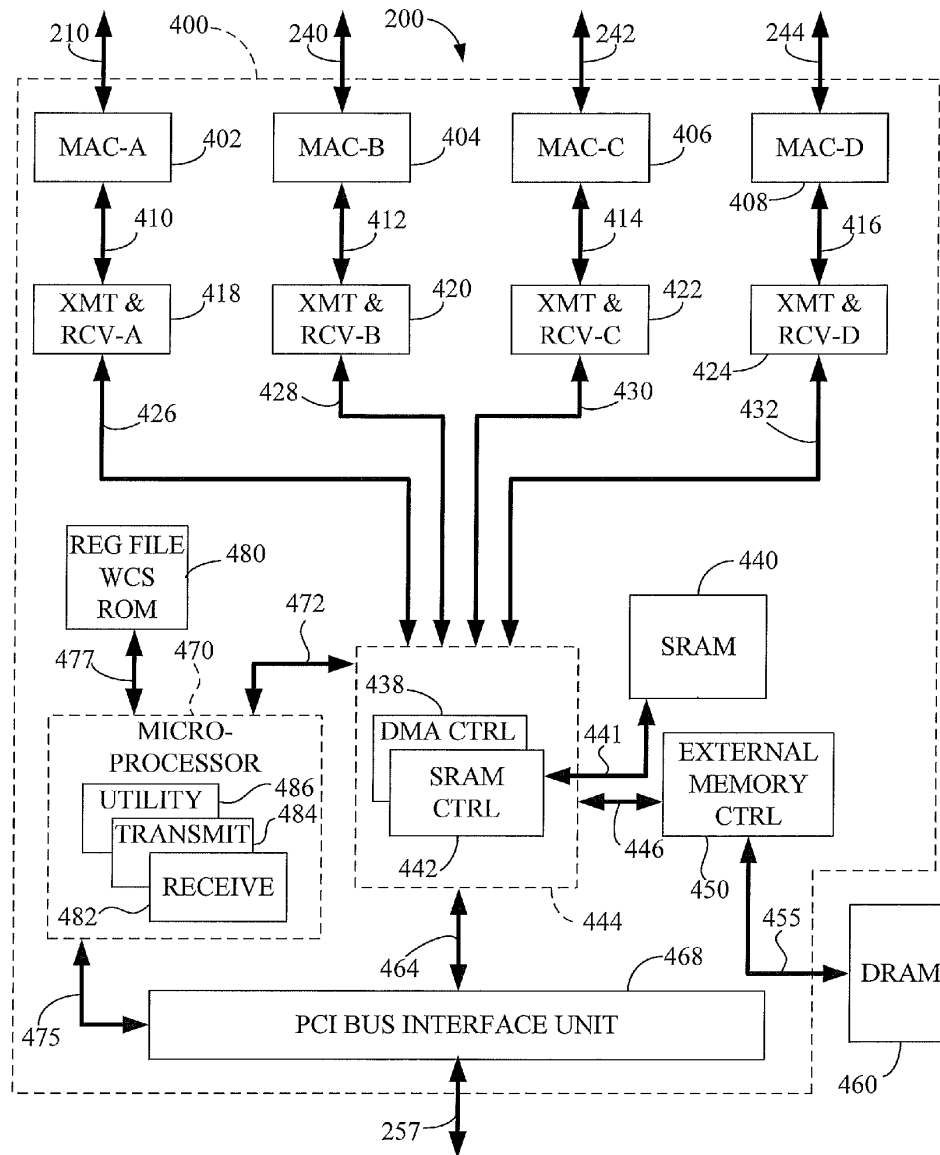


FIG. 13

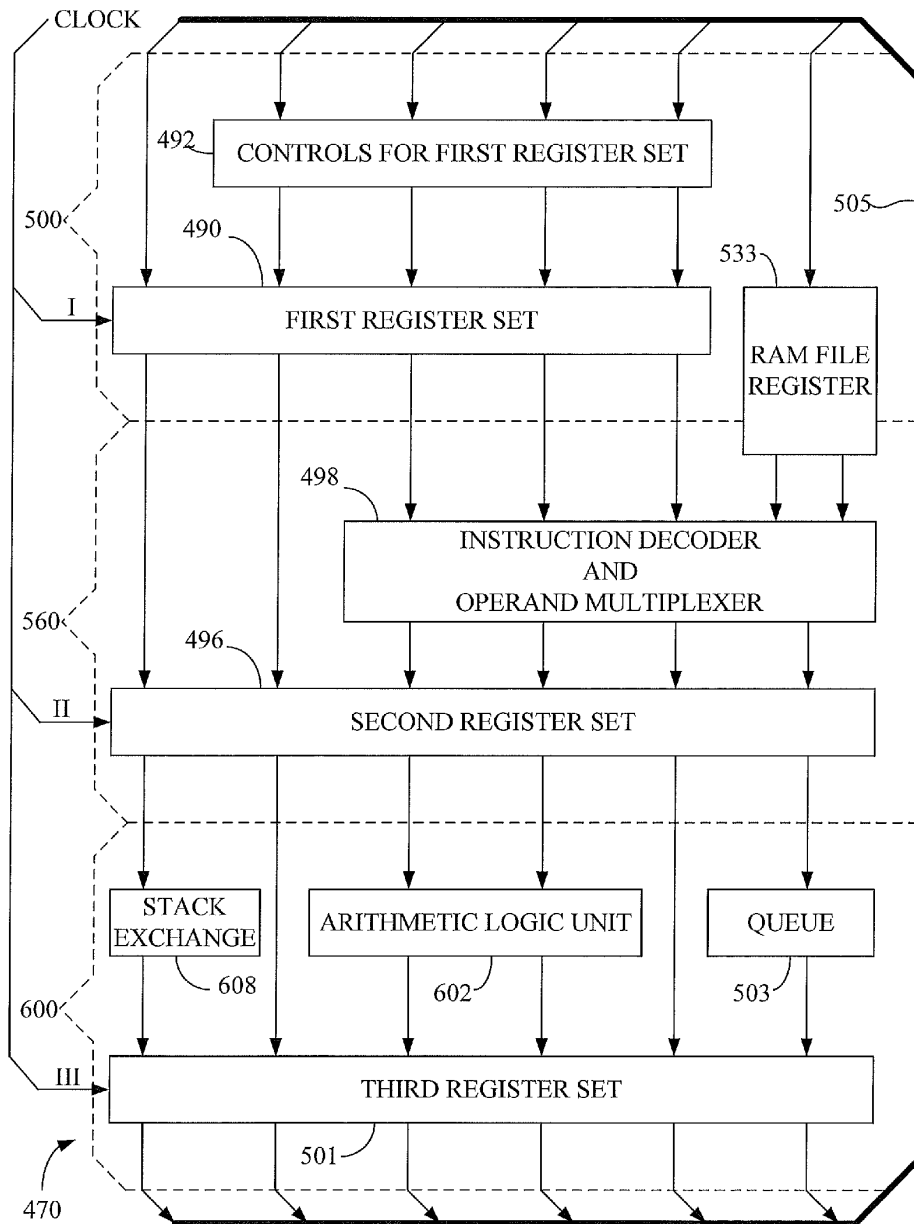


FIG. 14

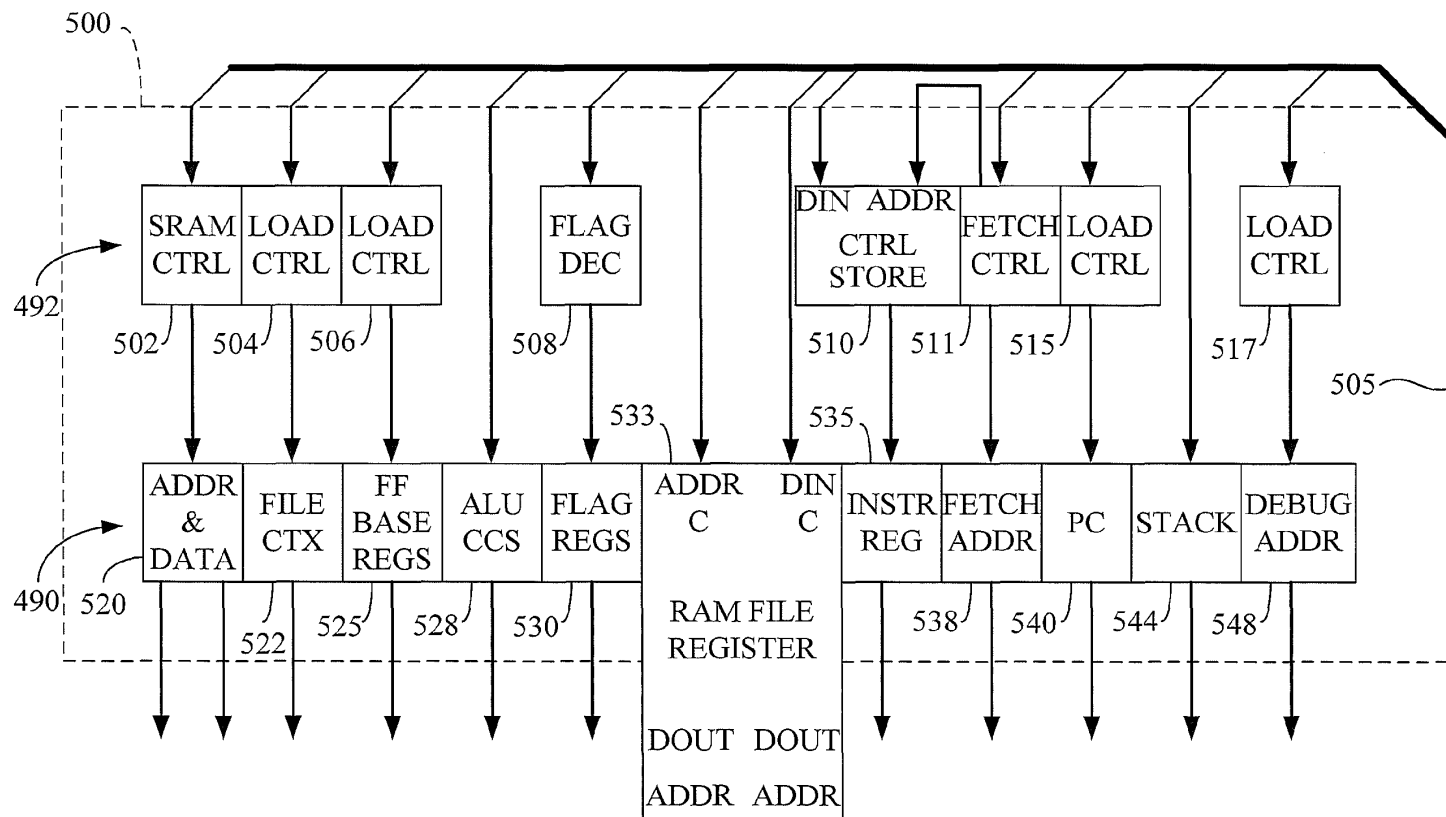


FIG. 15A

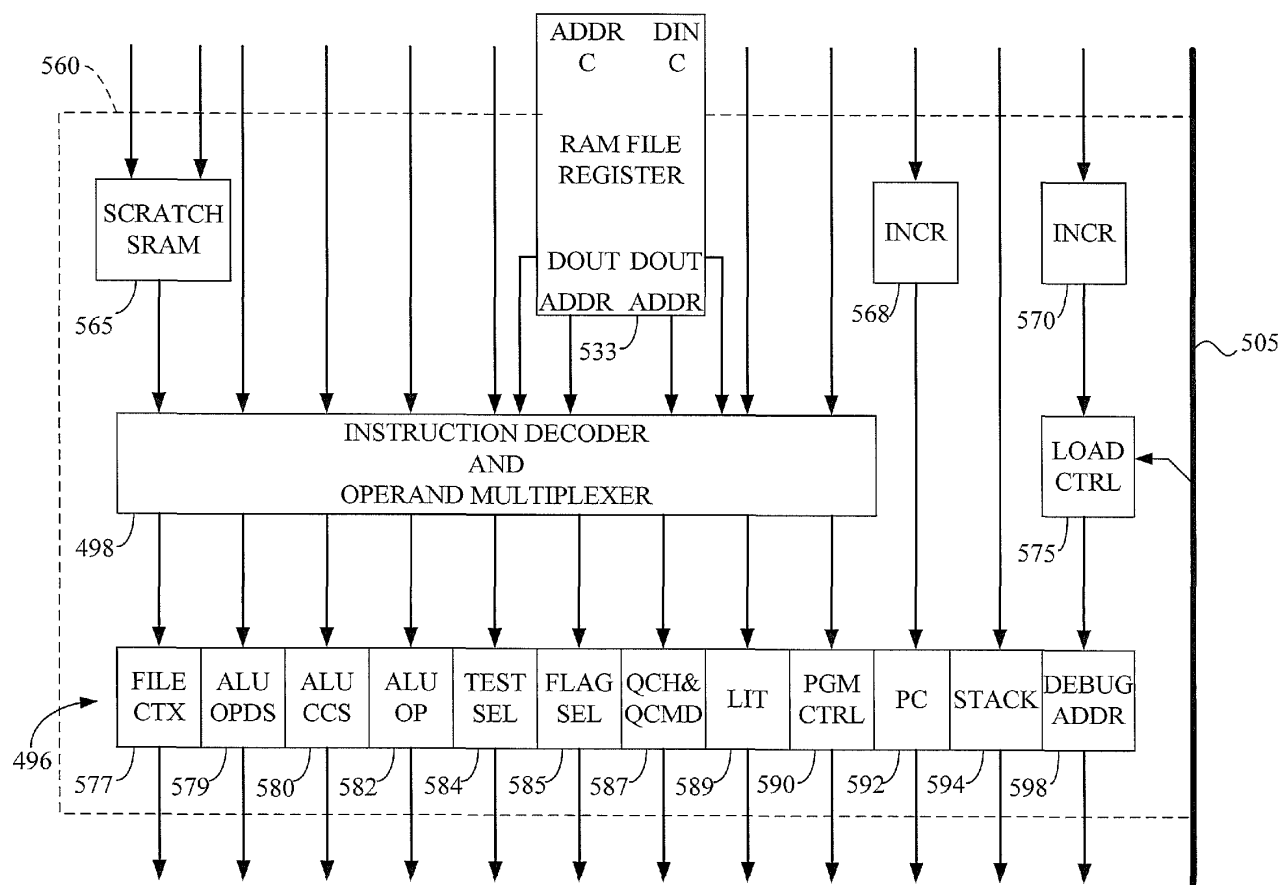


FIG. 15B

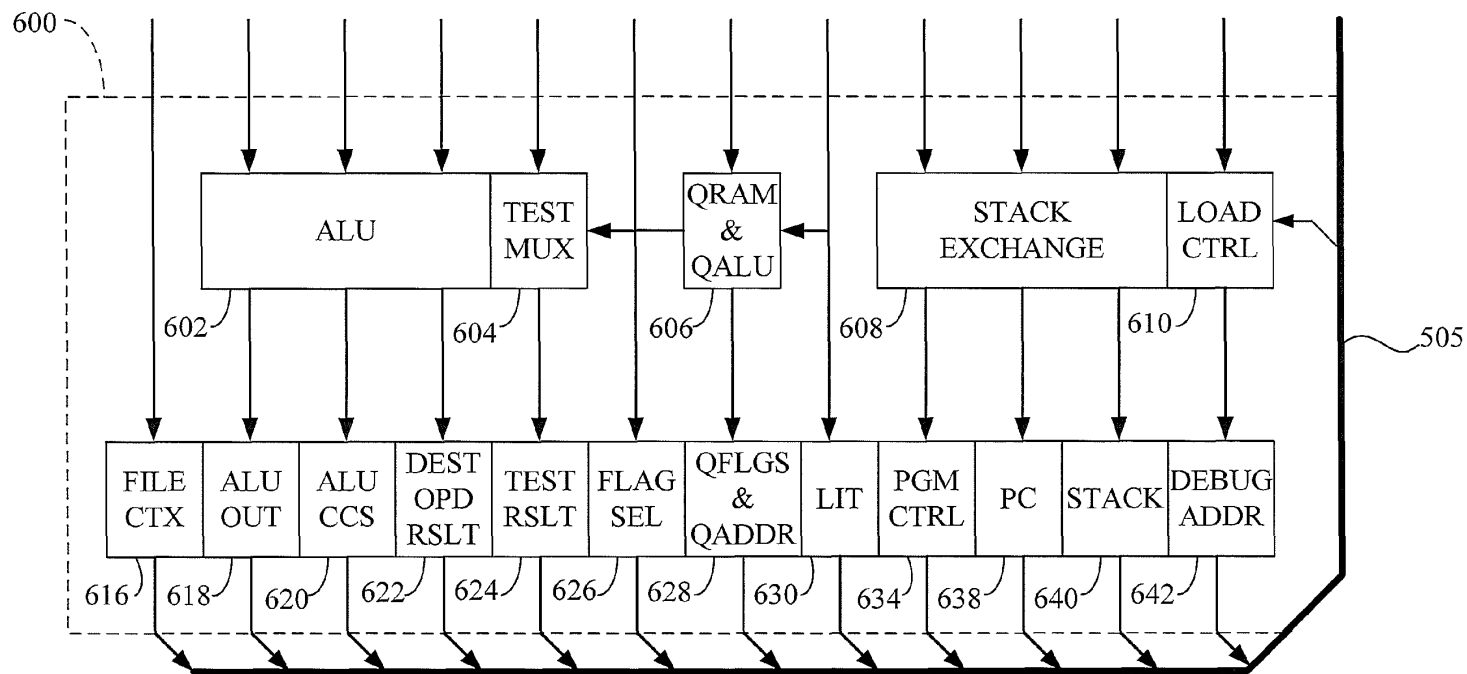


FIG. 15C

INTELLIGENT NETWORK INTERFACE SYSTEM AND METHOD FOR PROTOCOL PROCESSING

CROSS REFERENCE TO RELATED APPLICATIONS

This application claims the benefit under 35 U.S.C. §120 of (is a continuation of) U.S. patent application Ser. No. 09/692,561, filed Oct. 18, 2000, which in turn claims the benefit under 35 U.S.C. §120 of (is a continuation of) U.S. patent application Ser. No. 09/067,544, filed Apr. 28, 1998, now U.S. Pat. No. 6,226,680, which claims the benefit under 35 U.S.C. §119(e) of U.S. Patent Application Ser. No. 60/061,809, filed Oct. 14, 1997. The complete disclosure of all of the above applications is incorporated by reference herein.

TECHNICAL FIELD

The present invention relates generally to computer or other networks, and more particularly to protocol processing for information communicated between hosts such as computers connected to a network.

BACKGROUND

The advantages of network computing are increasingly evident. The convenience and efficiency of providing information, communication or computational power to individuals at their personal computer or other end user devices has led to rapid growth of such network computing, including internet as well as intranet systems and applications.

As is well known, most network computer communication is accomplished with the aid of a layered software architecture for moving information between host computers connected to the network. The layers help to segregate information into manageable segments, the general functions of each layer often based on an international standard called Open Systems Interconnection (OSI). OSI sets forth seven processing layers through which information may pass when received by a host in order to be presentable to an end user. Similarly, transmission of information from a host to the network may pass through those seven processing layers in reverse order. Each step of processing and service by a layer may include copying the processed information. Another reference model that is widely implemented, called TCP/IP (TCP stands for transport control protocol, while IP denotes internet protocol) essentially employs five of the seven layers of OSI.

Networks may include, for instance, a high-speed bus such as an Ethernet connection or an internet connection between disparate local area networks (LANs), each of which includes multiple hosts, or any of a variety of other known means for data transfer between hosts. According to the OSI standard, physical layers are connected to the network at respective hosts, the physical layers providing transmission and receipt of raw data bits via the network. A data link layer is serviced by the physical layer of each host, the data link layers providing frame division and error correction to the data received from the physical layers, as well as processing acknowledgment frames sent by the receiving host. A network layer of each host is serviced by respective data link layers, the network layers primarily controlling size and coordination of subnets of packets of data.

A transport layer is serviced by each network layer and a session layer is serviced by each transport layer within each host. Transport layers accept data from their respective ses-

sion layers and split the data into smaller units for transmission to the other host's transport layer, which concatenates the data for presentation to respective presentation layers. Session layers allow for enhanced communication control between the hosts. Presentation layers are serviced by their respective session layers, the presentation layers translating between data semantics and syntax which may be peculiar to each host and standardized structures of data representation. Compression and/or encryption of data may also be accomplished at the presentation level. Application layers are serviced by respective presentation layers, the application layers translating between programs particular to individual hosts and standardized programs for presentation to either an application or an end user. The TCP/IP standard includes the lower four layers and application layers, but integrates the functions of session layers and presentation layers into adjacent layers. Generally speaking, application, presentation and session layers are defined as upper layers, while transport, network and data link layers are defined as lower layers.

The rules and conventions for each layer are called the protocol of that layer, and since the protocols and general functions of each layer are roughly equivalent in various hosts, it is useful to think of communication occurring directly between identical layers of different hosts, even though these peer layers do not directly communicate without information transferring sequentially through each layer below. Each lower layer performs a service for the layer immediately above it to help with processing the communicated information. Each layer saves the information for processing and service to the next layer. Due to the multiplicity of hardware and software architectures, systems and programs commonly employed, each layer is necessary to insure that the data can make it to the intended destination in the appropriate form, regardless of variations in hardware and software that may intervene.

In preparing data for transmission from a first to a second host, some control data is added at each layer of the first host regarding the protocol of that layer, the control data being indistinguishable from the original (payload) data for all lower layers of that host. Thus an application layer attaches an application header to the payload data and sends the combined data to the presentation layer of the sending host, which receives the combined data, operates on it and adds a presentation header to the data, resulting in another combined data packet. The data resulting from combination of payload data, application header and presentation header is then passed to the session layer, which performs required operations including attaching a session header to the data and presenting the resulting combination of data to the transport layer. This process continues as the information moves to lower layers, with a transport header, network header and data link header and trailer attached to the data at each of those layers, with each step typically including data moving and copying, before sending the data as bit packets over the network to the second host.

The receiving host generally performs the converse of the above-described process, beginning with receiving the bits from the network, as headers are removed and data processed in order from the lowest (physical) layer to the highest (application) layer before transmission to a destination of the receiving host. Each layer of the receiving host recognizes and manipulates only the headers associated with that layer, since to that layer the higher layer control data is included with and indistinguishable from the payload data. Multiple interrupts, valuable central processing unit (CPU) processing

time and repeated data copies may also be necessary for the receiving host to place the data in an appropriate form at its intended destination.

The above description of layered protocol processing is simplified, as college-level textbooks devoted primarily to this subject are available, such as Computer Networks, Third Edition (1996) by Andrew S. Tanenbaum, which is incorporated herein by reference. As defined in that book, a computer network is an interconnected collection of autonomous computers, such as internet and intranet systems, including local area networks (LANs), wide area networks (WANs), asynchronous transfer mode (ATM), ring or token ring, wired, wireless, satellite or other means for providing communication capability between separate processors. A computer is defined herein to include a device having both logic and memory functions for processing data, while computers or hosts connected to a network are said to be heterogeneous if they function according to different operating systems or communicate via different architectures.

As networks grow increasingly popular and the information communicated thereby becomes increasingly complex and copious, the need for such protocol processing has increased. It is estimated that a large fraction of the processing power of a host CPU may be devoted to controlling protocol processes, diminishing the ability of that CPU to perform other tasks. Network interface cards have been developed to help with the lowest layers, such as the physical and data link layers. It is also possible to increase protocol processing speed by simply adding more processing power or CPUs according to conventional arrangements. This solution, however, is both awkward and expensive. But the complexities presented by various networks, protocols, architectures, operating systems and applications generally require extensive processing to afford communication capability between various network hosts.

SUMMARY OF THE INVENTION

The current invention provides a system for processing network communication that greatly increases the speed of that processing and the efficiency of moving the data being communicated. The invention has been achieved by questioning the long-standing practice of performing multilayered protocol processing on a general-purpose processor. The protocol processing method and architecture that results effectively collapses the layers of a connection-based, layered architecture such as TCP/IP into a single wider layer which is able to send network data more directly to and from a desired location or buffer on a host. This accelerated processing is provided to a host for both transmitting and receiving data, and so improves performance whether one or both hosts involved in an exchange of information have such a feature.

The accelerated processing includes employing representative control instructions for a given message that allow data from the message to be processed via a fast-path which accesses message data directly at its source or delivers it directly to its intended destination. This fast-path bypasses conventional protocol processing of headers that accompany the data. The fast-path employs a specialized microprocessor designed for processing network communication, avoiding the delays and pitfalls of conventional software layer processing, such as repeated copying and interrupts to the CPU. In effect, the fast-path replaces the states that are traditionally found in several layers of a conventional network stack with a single state machine encompassing all those layers, in contrast to conventional rules that require rigorous differentiation and separation of protocol layers. The host retains a sequen-

tial protocol processing stack which can be employed for setting up a fast-path connection or processing message exceptions. The specialized microprocessor and the host intelligently choose whether a given message or portion of a message is processed by the microprocessor or the host stack.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a plan view diagram of a system of the present invention, including a host computer having a communication-processing device for accelerating network communication.

FIG. 2 is a diagram of information flow for the host of FIG. 1 in processing network communication, including a fast-path, a slow-path and a transfer of connection context between the fast and slow-paths.

FIG. 3 is a flow chart of message receiving according to the present invention.

FIG. 4A is a diagram of information flow for the host of FIG. 1 receiving a message packet processed by the slow-path.

FIG. 4B is a diagram of information flow for the host of FIG. 1 receiving an initial message packet processed by the fast-path.

FIG. 4C is a diagram of information flow for the host of FIG. 4B receiving a subsequent message packet processed by the fast-path.

FIG. 4D is a diagram of information flow for the host of FIG. 4C receiving a message packet having an error that causes processing to revert to the slow-path.

FIG. 5 is a diagram of information flow for the host of FIG. 1 transmitting a message by either the fast or slow-paths.

FIG. 6 is a diagram of information flow for a first embodiment of an intelligent network interface card (INIC) associated with a client having a TCP/IP processing stack.

FIG. 7 is a diagram of hardware logic for the INIC embodiment shown in FIG. 6, including a packet control sequencer and a fly-by sequencer.

FIG. 8 is a diagram of the fly-by sequencer of FIG. 7 for analyzing header bytes as they are received by the INIC.

FIG. 9 is a diagram of information flow for a second embodiment of an INIC associated with a server having a TCP/IP processing stack.

FIG. 10 is a diagram of a command driver installed in the host of FIG. 9 for creating and controlling a communication control block for the fast-path.

FIG. 11 is a diagram of the TCP/IP stack and command driver of FIG. 10 configured for NetBios communications.

FIG. 12 is a diagram of a communication exchange between the client of FIG. 6 and the server of FIG. 9.

FIG. 13 is a diagram of hardware functions included in the INIC of FIG. 9.

FIG. 14 is a diagram of a trio of pipelined microprocessors included in the INIC of FIG. 13, including three phases with a processor in each phase.

FIG. 15A is a diagram of a first phase of the pipelined microprocessor of FIG. 14.

FIG. 15B is a diagram of a second phase of the pipelined microprocessor of FIG. 14.

FIG. 15C is a diagram of a third phase of the pipelined microprocessor of FIG. 14.

DETAILED DESCRIPTION

FIG. 1 shows a host 20 of the present invention connected by a network 25 to a remote host 22. The increase in processing speed achieved by the present invention can be provided

5

with an intelligent network interface card (INIC) that is easily and affordably added to an existing host, or with a communication processing device (CPD) that is integrated into a host, in either case freeing the host CPU from most protocol processing and allowing improvements in other tasks performed by that CPU. The host 20 in a first embodiment contains a CPU 28 and a CPD 30 connected by a host bus 33. The CPD 30 includes a microprocessor designed for processing communication data and memory buffers controlled by a direct memory access (DMA) unit. Also connected to the host bus 33 is a storage device 35, such as a semiconductor memory or disk drive, along with any related controls.

Referring additionally to FIG. 2, the host CPU 28 controls a protocol processing stack 44 housed in storage 35, the stack including a data link layer 36, network layer 38, transport layer 40, upper layer 46 and an upper layer interface 42. The upper layer 46 may represent a session, presentation and/or application layer, depending upon the particular protocol being employed and message communicated. The upper layer interface 42, along with the CPU 28 and any related controls can send or retrieve a file to or from the upper layer 46 or storage 35, as shown by arrow 48. A connection context 50 has been created, as will be explained below, the context summarizing various features of the connection, such as protocol type and source and destination addresses for each protocol layer. The context may be passed between an interface for the session layer 42 and the CPD 30, as shown by arrows 52 and 54, and stored as a communication control block (CCB) at either CPD 30 or storage 35.

When the CPD 30 holds a CCB defining a particular connection, data received by the CPD from the network and pertaining to the connection is referenced to that CCB and can then be sent directly to storage 35 according to a fast-path 58, bypassing sequential protocol processing by the data link 36, network 38 and transport 40 layers. Transmitting a message, such as sending a file from storage 35 to remote host 22, can also occur via the fast-path 58, in which case the context for the file data is added by the CPD 30 referencing a CCB, rather than by sequentially adding headers during processing by the transport 40, network 38 and data link 36 layers. The DMA controllers of the CPD 30 perform these transfers between CPD and storage 35.

The CPD 30 collapses multiple protocol stacks each having possible separate states into a single state machine for fast-path processing. As a result, exception conditions may occur that are not provided for in the single state machine, primarily because such conditions occur infrequently and to deal with them on the CPD would provide little or no performance benefit to the host. Such exceptions can be CPD 30 or CPU 28 initiated. An advantage of the invention includes the manner in which unexpected situations that occur on a fast-path CCB are handled. The CPD 30 deals with these rare situations by passing back or flushing to the host protocol stack 44 the CCB and any associated message frames involved, via a control negotiation. The exception condition is then processed in a conventional manner by the host protocol stack 44. At some later time, usually directly after the handling of the exception condition has completed and fast-path processing can resume, the host stack 44 hands the CCB back to the CPD.

This fallback capability enables the performance-impacting functions of the host protocols to be handled by the CPD network microprocessor, while the exceptions are dealt with by the host stacks, the exceptions being so rare as to negligibly effect overall performance. The custom designed network microprocessor can have independent processors for transmitting and receiving network information, and further pro-

6

cessors for assisting and queuing. A preferred microprocessor embodiment includes a pipelined trio of receive, transmit and utility processors. DMA controllers are integrated into the implementation and work in close concert with the network microprocessor to quickly move data between buffers adjacent the controllers and other locations such as long term storage. Providing buffers logically adjacent to the DMA controllers avoids unnecessary loads on the PCI bus.

FIG. 3 diagrams the general flow of messages received according to the current invention. A large TCP/IP message such as a file transfer may be received by the host from the network in a number of separate, approximately 64 KB transfers, each of which may be split into many, approximately 1.5 KB frames or packets for transmission over a network. Novel NetWare protocol suites running Sequenced Packet Exchange Protocol (SPX) or NetWare Core Protocol (NCP) over Internetwork Packet Exchange (IPX) work in a similar fashion. Another form of data communication which can be handled by the fast-path is Transaction TCP (hereinafter T/TCP or TTCP), a version of TCP which initiates a connection with an initial transaction request after which a reply containing data may be sent according to the connection, rather than initiating a connection via a several-message initialization dialogue and then transferring data with later messages. In any of the transfers typified by these protocols, each packet conventionally includes a portion of the data being transferred, as well as headers for each of the protocol layers and markers for positioning the packet relative to the rest of the packets of this message.

When a message packet or frame is received 47 from a network by the CPD, it is first validated by a hardware assist. This includes determining the protocol types of the various layers, verifying relevant checksums, and summarizing 57 these findings into a status word or words. Included in these words is an indication whether or not the frame is a candidate for fast-path data flow. Selection 59 of fast-path candidates is based on whether the host may benefit from this message connection being handled by the CPD, which includes determining whether the packet has header bytes denoting particular protocols, such as TCP/IP or SPX/IPX for example. The small percent of frames that are not fast-path candidates are sent 61 to the host protocol stacks for slow-path protocol processing. Subsequent network microprocessor work with each fast-path candidate determines whether a fast-path connection such as a TCP or SPX CCB is already extant for that candidate, or whether that candidate may be used to set up a new fast-path connection, such as for a TTCP/IP transaction. The validation provided by the CPD provides acceleration whether a frame is processed by the fast-path or a slow-path, as only error free, validated frames are processed by the host CPU even for the slow-path processing.

All received message frames which have been determined by the CPD hardware assist to be fast-path candidates are examined 53 by the network microprocessor or INIC comparator circuits to determine whether they match a CCB held by the CPD. Upon confirming such a match, the CPD removes lower layer headers and sends 69 the remaining application data from the frame directly into its final destination in the host using direct memory access (DMA) units of the CPD. This operation may occur immediately upon receipt of a message packet, for example when a TCP connection already exists and destination buffers have been negotiated, or it may first be necessary to process an initial header to acquire a new set of final destination addresses for this transfer. In this latter case, the CPD will queue subsequent message packets while waiting for the destination address, and then DMA the queued application data to that destination.

A fast-path candidate that does not match a CCB may be used to set up a new fast-path connection, by sending 65 the frame to the host for sequential protocol processing. In this case, the host uses this frame to create 51 a CCB, which is then passed to the CPD to control subsequent frames on that connection. The CCB, which is cached 67 in the CPD, includes control and state information pertinent to all protocols that would have been processed had conventional software layer processing been employed. The CCB also contains storage space for per-transfer information used to facilitate moving application-level data contained within subsequent related message packets directly to a host application in a form available for immediate usage. The CPD takes command of connection processing upon receiving a CCB for that connection from the host.

As shown more specifically in FIG. 4A, when a message packet is received from the remote host 22 via network 25, the packet enters hardware receive logic 32 of the CPD 30, which checks headers and data, and parses the headers, creating a word or words which identify the message packet and status, storing the headers, data and word temporarily in memory 60. As well as validating the packet, the receive logic 32 indicates with the word whether this packet is a candidate for fast-path processing. FIG. 4A depicts the case in which the packet is not a fast-path candidate, in which case the CPD 30 sends the validated headers and data from memory 60 to data link layer 36 along an internal bus for processing by the host CPU, as shown by arrow 56. The packet is processed by the host protocol stack 44 of data link 36, network 38, transport 40 and session 42 layers, and data (D) 63 from the packet may then be sent to storage 35, as shown by arrow 65.

FIG. 4B, depicts the case in which the receive logic 32 of the CPD determines that a message packet is a candidate for fast-path processing, for example by deriving from the packet's headers that the packet belongs to a TCP/IP, TTCP/IP or SPX/IPX message. A processor 55 in the CPD 30 then checks to see whether the word that summarizes the fast-path candidate matches a CCB held in a cache 62. Upon finding no match for this packet, the CPD sends the validated packet from memory 60 to the host protocol stack 44 for processing. Host stack 44 may use this packet to create a connection context for the message, including finding and reserving a destination for data from the message associated with the packet, the context taking the form of a CCB. The present embodiment employs a single specialized host stack 44 for processing both fast-path and non-fast-path candidates, while in an embodiment described below fast-path candidates are processed by a different host stack than non-fast-path candidates. Some data (D1) 66 from that initial packet may optionally be sent to the destination in storage 35, as shown by arrow 68. The CCB is then sent to the CPD 30 to be saved in cache 62, as shown by arrow 64. For a traditional connection-based message such as typified by TCP/IP, the initial packet may be part of a connection initialization dialogue that transpires between hosts before the CCB is created and passed to the CPD 30.

Referring now to FIG. 4C, when a subsequent packet from the same connection as the initial packet is received from the network 25 by CPD 30, the packet headers and data are validated by the receive logic 32, and the headers are parsed to create a summary of the message packet and a hash for finding a corresponding CCB, the summary and hash contained in a word or words. The word or words are temporarily stored in memory 60 along with the packet. The processor 55 checks for a match between the hash and each CCB that is stored in the cache 62 and, finding a match, sends the data (D2) 70 via a fast-path directly to the destination in storage

35, as shown by arrow 72, bypassing the session layer 42, transport layer 40, network layer 38 and data link layer 36. The remaining data packets from the message can also be sent by DMA directly to storage, avoiding the relatively slow protocol layer processing and repeated copying by the CPU stack 44.

FIG. 4D shows the procedure for handling the rare instance when a message for which a fast-path connection has been established, such as shown in FIG. 4C, has a packet that is not easily handled by the CPD. In this case the packet is sent to be processed by the protocol stack 44, which is handed the CCB for that message from cache 62 via a control dialogue with the CPD, as shown by arrow 76, signaling to the CPU to take over processing of that message. Slow-path processing by the protocol stack then results in data (D3) 80 from the packet being sent, as shown by arrow 82, to storage 35. Once the packet has been processed and the error situation corrected, the CCB can be handed back via a control dialogue to the cache 62, so that payload data from subsequent packets of that message can again be sent via the fast-path of the CPD 30. Thus the CPU and CPD together decide whether a given message is to be processed according to fast-path hardware processing or more conventional software processing by the CPU.

Transmission of a message from the host 20 to the network 25 for delivery to remote host 22 also can be processed by either sequential protocol software processing via the CPU or accelerated hardware processing via the CPD 30, as shown in FIG. 5. A message (M) 90 that is selected by CPU 28 from storage 35 can be sent to session layer 42 for processing by stack 44, as shown by arrows 92 and 96. For the situation in which a connection exists and the CPD 30 already has an appropriate CCB for the message, however, data packets can bypass host stack 44 and be sent by DMA directly to memory 60, with the processor 55 adding to each data packet a single header containing all the appropriate protocol layers, and sending the resulting packets to the network 25 for transmission to remote host 22. This fast-path transmission can greatly accelerate processing for even a single packet, with the acceleration multiplied for a larger message.

A message for which a fast-path connection is not extant thus may benefit from creation of a CCB with appropriate control and state information for guiding fast-path transmission. For a traditional connection-based message, such as typified by TCP/IP or SPX/IPX, the CCB is created during connection initialization dialogue. For a quick-connection message, such as typified by TTCP/IP, the CCB can be created with the same transaction that transmits payload data. In this case, the transmission of payload data may be a reply to a request that was used to set up the fast-path connection. In any case, the CCB provides protocol and status information regarding each of the protocol layers, including which user is involved and storage space for per-transfer information. The CCB is created by protocol stack 44, which then passes the CCB to the CPD 30 by writing to a command register of the CPD, as shown by arrow 98. Guided by the CCB, the processor 55 moves network frame-sized portions of the data from the source in host memory 35 into its own memory 60 using DMA, as depicted by arrow 99. The processor 55 then prepends appropriate headers and checksums to the data portions, and transmits the resulting frames to the network 25, consistent with the restrictions of the associated protocols. After the CPD 30 has received an acknowledgement that all the data has reached its destination, the CPD will then notify the host 35 by writing to a response buffer.

Thus, fast-path transmission of data communications also relieves the host CPU of per-frame processing. A vast majority of data transmissions can be sent to the network by the

fast-path. Both the input and output fast-paths attain a huge reduction in interrupts by functioning at an upper layer level, i.e., session level or higher, and interactions between the network microprocessor and the host occur using the full transfer sizes which that upper layer wishes to make. For fast-path communications, an interrupt only occurs (at the most) at the beginning and end of an entire upper-layer message transaction, and there are no interrupts for the sending or receiving of each lower layer portion or packet of that transaction.

A simplified intelligent network interface card (INIC) 150 is shown in FIG. 6 to provide a network interface for a host 152. Hardware logic 171 of the INIC 150 is connected to a network 155, with a peripheral bus (PCI) 157 connecting the INIC and host. The host 152 in this embodiment has a TCP/IP protocol stack, which provides a slow-path 158 for sequential software processing of message frames received from the network 155. The host 152 protocol stack includes a data link layer 160, network layer 162, a transport layer 164 and an application layer 166, which provides a source or destination 168 for the communication data in the host 152. Other layers which are not shown, such as session and presentation layers, may also be included in the host stack 152, and the source or destination may vary depending upon the nature of the data and may actually be the application layer.

The INIC 150 has a network processor 170 which chooses between processing messages along a slow-path 158 that includes the protocol stack of the host, or along a fast-path 159 that bypasses the protocol stack of the host. Each received packet is processed on the fly by hardware logic 171 contained in INIC 150, so that all of the protocol headers for a packet can be processed without copying, moving or storing the data between protocol layers. The hardware logic 171 processes the headers of a given packet at one time as packet bytes pass through the hardware, by categorizing selected header bytes. Results of processing the selected bytes help to determine which other bytes of the packet are categorized, until a summary of the packet has been created, including checksum validations. The processed headers and data from the received packet are then stored in INIC storage 185, as well as the word or words summarizing the headers and status of the packet.

The hardware processing of message packets received by INIC 150 from network 155 is shown in more detail in FIG. 7. A received message packet first enters a media access controller 172, which controls INIC access to the network and receipt of packets and can provide statistical information for network protocol management. From there, data flows one byte at a time into an assembly register 174, which in this example is 128 bits wide. The data is categorized by a fly-by sequencer 178, as will be explained in more detail with regard to FIG. 8, which examines the bytes of a packet as they fly by, and generates status from those bytes that will be used to summarize the packet. The status thus created is merged with the data by a multiplexer 180 and the resulting data stored in SRAM 182. A packet control sequencer 176 oversees the fly-by sequencer 178, examines information from the media access controller 172, counts the bytes of data, generates addresses, moves status and manages the movement of data from the assembly register 174 to SRAM 182 and eventually DRAM 188. The packet control sequencer 176 manages a buffer in SRAM 182 via SRAM controller 183, and also indicates to a DRAM controller 186 when data needs to be moved from SRAM 182 to a buffer in DRAM 188. Once data movement for the packet has been completed and all the data has been moved to the buffer in DRAM 188, the packet control sequencer 176 will move the status that has been

generated in the fly-by sequencer 178 out to the SRAM 182 and to the beginning of the DRAM 188 buffer to be prepended to the packet data. The packet control sequencer 176 then requests a queue manager 184 to enter a receive buffer descriptor into a receive queue, which in turn notifies the processor 170 that the packet has been processed by hardware logic 171 and its status summarized.

FIG. 8 shows that the fly-by sequencer 178 has several tiers, with each tier generally focusing on a particular portion of the packet header and thus on a particular protocol layer, for generating status pertaining to that layer. The fly-by sequencer 178 in this embodiment includes a media access control sequencer 191, a network sequencer 192, a transport sequencer 194 and a session sequencer 195. Sequencers pertaining to higher protocol layers can additionally be provided. The fly-by sequencer 178 is reset by the packet control sequencer 176 and given pointers by the packet control sequencer that tell the fly-by sequencer whether a given byte is available from the assembly register 174. The media access control sequencer 191 determines, by looking at bytes 0-5, that a packet is addressed to host 152 rather than or in addition to another host. Offsets 12 and 13 of the packet are also processed by the media access control sequencer 191 to determine the type field, for example whether the packet is Ethernet or 802.3. If the type field is Ethernet those bytes also tell the media access control sequencer 191 the packet's network protocol type. For the 802.3 case, those bytes instead indicate the length of the entire frame, and the media access control sequencer 191 will check eight bytes further into the packet to determine the network layer type.

For most packets the network sequencer 192 validates that the header length received has the correct length, and checksums the network layer header. For fast-path candidates the network layer header is known to be IP or IPX from analysis done by the media access control sequencer 191. Assuming for example that the type field is 802.3 and the network protocol is IP, the network sequencer 192 analyzes the first bytes of the network layer header, which will begin at byte 22, in order to determine IP type. The first bytes of the IP header will be processed by the network sequencer 192 to determine what IP type the packet involves. Determining that the packet involves, for example, IP version 4, directs further processing by the network sequencer 192, which also looks at the protocol type located ten bytes into the IP header for an indication of the transport header protocol of the packet. For example, for IP over Ethernet, the IP header begins at offset 14, and the protocol type byte is offset 23, which will be processed by network logic to determine whether the transport layer protocol is TCP, for example. From the length of the network layer header, which is typically 20-40 bytes, network sequencer 192 determines the beginning of the packet's transport layer header for validating the transport layer header. Transport sequencer 194 may generate checksums for the transport layer header and data, which may include information from the IP header in the case of TCP at least.

Continuing with the example of a TCP packet, transport sequencer 194 also analyzes the first few bytes in the transport layer portion of the header to determine, in part, the TCP source and destination ports for the message, such as whether the packet is NetBios or other protocols. Byte 12 of the TCP header is processed by the transport sequencer 194 to determine and validate the TCP header length. Byte 13 of the TCP header contains flags that may, aside from ack flags and push flags, indicate unexpected options, such as reset and fin, that may cause the processor to categorize this packet as an exception. TCP offset bytes 16 and 17 are the checksum, which is

11

pulled out and stored by the hardware logic 171 while the rest of the frame is validated against the checksum.

Session sequencer 195 determines the length of the session layer header, which in the case of NetBios is only four bytes, two of which tell the length of the NetBios payload data, but which can be much larger for other protocols. The session sequencer 195 can also be used to categorize the type of message as read or write, for example, for which the fast-path may be particularly beneficial. Further upper layer logic processing, depending upon the message type, can be performed by the hardware logic 171 of packet control sequencer 176 and fly-by sequencer 178. Thus hardware logic 171 intelligently directs hardware processing of the headers by categorization of selected bytes from a single stream of bytes, with the status of the packet being built from classifications determined on the fly. Once the packet control sequencer 176 detects that all of the packet has been processed by the fly-by sequencer 178, the packet control sequencer 176 adds the status information generated by the fly-by sequencer 178 and any status information generated by the packet control sequencer 176, and prepends (adds to the front) that status information to the packet, for convenience in handling the packet by the processor 170. The additional status information generated by the packet control sequencer 176 includes media access controller 172 status information and any errors discovered, or data overflow in either the assembly register or DRAM buffer, or other miscellaneous information regarding the packet. The packet control sequencer 176 also stores entries into a receive buffer queue and a receive statistics queue via the queue manager 184.

An advantage of processing a packet by hardware logic 171 is that the packet does not, in contrast with conventional sequential software protocol processing, have to be stored, moved, copied or pulled from storage for processing each protocol layer header, offering dramatic increases in processing efficiency and savings in processing time for each packet. The packets can be processed at the rate bits are received from the network, for example 100 megabits/second for a 100 baseT connection. The time for categorizing a packet received at this rate and having a length of sixty bytes is thus about 5 microseconds. The total time for processing this packet with the hardware logic 171 and sending packet data to its host destination via the fast-path may be about 16 microseconds or less, assuming a 66 MHz PCI bus, whereas conventional software protocol processing by a 300 MHz Pentium II® processor may take as much as 200 microseconds in a busy system. More than an order of magnitude decrease in processing time can thus be achieved with fast-path 159 in comparison with a high-speed CPU employing conventional sequential software protocol processing, demonstrating the dramatic acceleration provided by processing the protocol headers by the hardware logic 171 and processor 170, without even considering the additional time savings afforded by the reduction in CPU interrupts and host bus bandwidth savings.

The processor 170 chooses, for each received message packet held in storage 185, whether that packet is a candidate for the fast-path 159 and, if so, checks to see whether a fast-path has already been set up for the connection that the packet belongs to. To do this, the processor 170 first checks the header status summary to determine whether the packet headers are of a protocol defined for fast-path candidates. If not, the processor 170 commands DMA controllers in the INIC 150 to send the packet to the host for slow-path 158 processing. Even for a slow-path 158 processing of a message, the INIC 150 thus performs initial procedures such as

12

validation and determination of message type, and passes the validated message at least to the data link layer 160 of the host.

For fast-path 159 candidates, the processor 170 checks to see whether the header status summary matches a CCB held by the INIC. If so, the data from the packet is sent along fast-path 159 to the destination 168 in the host. If the fast-path 159 candidate's packet summary does not match a CCB held by the INIC, the packet may be sent to the host 152 for slow-path processing to create a CCB for the message. Employment of the fast-path 159 may also not be needed or desirable for the case of fragmented messages or other complexities. For the vast majority of messages, however, the INIC fast-path 159 can greatly accelerate message processing. The INIC 150 thus provides a single state machine processor 170 that decides whether to send data directly to its destination, based upon information gleaned on the fly, as opposed to the conventional employment of a state machine in each of several protocol layers for determining the destiny of a given packet.

In processing an indication or packet received at the host 152, a protocol driver of the host selects the processing route based upon whether the indication is fast-path or slow-path. A TCP/IP or SPX/IPX message has a connection that is set up from which a CCB is formed by the driver and passed to the INIC for matching with and guiding the fast-path packet to the connection destination 168. For a TTCP/IP message, the driver can create a connection context for the transaction from processing an initial request packet, including locating the message destination 168, and then passing that context to the INIC in the form of a CCB for providing a fast-path for a reply from that destination. A CCB includes connection and state information regarding the protocol layers and packets of the message. Thus a CCB can include source and destination media access control (MAC) addresses, source and destination IP or IPX addresses, source and destination TCP or SPX ports, TCP variables such as timers, receive and transmit windows for sliding window protocols, and information denoting the session layer protocol.

Caching the CCBs in a hash table in the INIC provides quick comparisons with words summarizing incoming packets to determine whether the packets can be processed via the fast-path 159, while the full CCBs are also held in the INIC for processing. Other ways to accelerate this comparison include software processes such as a B-tree or hardware assists such as a content addressable memory (CAM). When INIC microcode or comparator circuits detect a match with the CCB, a DMA controller places the data from the packet in the destination 168, without any interrupt by the CPU, protocol processing or copying. Depending upon the type of message received, the destination of the data may be the session, presentation or application layers, or a file buffer cache in the host 152.

FIG. 9 shows an INIC 200 connected to a host 202 that is employed as a file server. This INIC provides a network interface for several network connections employing the 802.3u standard, commonly known as Fast Ethernet. The INIC 200 is connected by a PCI bus 205 to the server 202, which maintains a TCP/IP or SPX/IPX protocol stack including MAC layer 212, network layer 215, transport layer 217 and application layer 220, with a source/destination 222 shown above the application layer, although as mentioned earlier the application layer can be the source or destination. The INIC is also connected to network lines 210, 240, 242 and 244, which are preferably fast Ethernet, twisted pair, fiber optic, coaxial cable or other lines each allowing data transmission of 100 Mb/s, while faster and slower data rates are

13

also possible. Network lines **210**, **240**, **242** and **244** are each connected to a dedicated row of hardware circuits which can each validate and summarize message packets received from their respective network line. Thus line **210** is connected with a first horizontal row of sequencers **250**, line **240** is connected with a second horizontal row of sequencers **260**, line **242** is connected with a third horizontal row of sequencers **262** and line **244** is connected with a fourth horizontal row of sequencers **264**. After a packet has been validated and summarized by one of the horizontal hardware rows it is stored along with its status summary in storage **270**.

A network processor **230** determines, based on that summary and a comparison with any CCBs stored in the INIC **200**, whether to send a packet along a slow-path **231** for processing by the host. A large majority of packets can avoid such sequential processing and have their data portions sent by DMA along a fast-path **237** directly to the data destination **222** in the server according to a matching CCB. Similarly, the fast-path **237** provides an avenue to send data directly from the source **222** to any of the network lines by processor **230** division of the data into packets and addition of full headers for network transmission, again minimizing CPU processing and interrupts. For clarity only horizontal sequencer **250** is shown active; in actuality each of the sequencer rows **250**, **260**, **262** and **264** offers full duplex communication, concurrently with all other sequencer rows. The specialized INIC **200** is much faster at working with message packets than even advanced general-purpose host CPUs that processes those headers sequentially according to the software protocol stack.

One of the most commonly used network protocols for large messages such as file transfers is server message block (SMB) over TCP/IP. SMB can operate in conjunction with redirector software that determines whether a required resource for a particular operation, such as a printer or a disk upon which a file is to be written, resides in or is associated with the host from which the operation was generated or is located at another host connected to the network, such as a file server. SMB and server/redirector are conventionally serviced by the transport layer; in the present invention SMB and redirector can instead be serviced by the INIC. In this case, sending data by the DMA controllers from the INIC buffers when receiving a large SMB transaction may greatly reduce interrupts that the host must handle. Moreover, this DMA generally moves the data to its final destination in the file system cache. An SMB transmission of the present invention follows essentially the reverse of the above described SMB receive, with data transferred from the host to the INIC and stored in buffers, while the associated protocol headers are prepended to the data in the INIC, for transmission via a network line to a remote host. Processing by the INIC of the multiple packets and multiple TCP, IP, NetBios and SMB protocol layers via custom hardware and without repeated interrupts of the host can greatly increase the speed of transmitting an SMB message to a network line.

As shown in FIG. 10, for controlling whether a given message is processed by the host **202** or by the INIC **200**, a message command driver **300** may be installed in host **202** to work in concert with a host protocol stack **310**. The command driver **300** can intervene in message reception or transmittal, create CCBs and send or receive CCBs from the INIC **200**, so that functioning of the INIC, aside from improved performance, is transparent to a user. Also shown is an INIC memory **304** and an INIC miniport driver **306**, which can direct message packets received from network **210** to either the conventional protocol stack **310** or the command protocol stack **300**, depending upon whether a packet has been labeled as a fast-path candidate. The conventional protocol stack **310**

14

has a data link layer **312**, a network layer **314** and a transport layer **316** for conventional, lower layer processing of messages that are not labeled as fast-path candidates and therefore not processed by the command stack **300**. Residing above the lower layer stack **310** is an upper layer **318**, which represents a session, presentation and/or application layer, depending upon the message communicated. The command driver **300** similarly has a data link layer **320**, a network layer **322** and a transport layer **325**.

The driver **300** includes an upper layer interface **330** that determines, for transmission of messages to the network **210**, whether a message transmitted from the upper layer **318** is to be processed by the command stack **300** and subsequently the INIC fast-path, or by the conventional stack **310**. When the upper layer interface **330** receives an appropriate message from the upper layer **318** that would conventionally be intended for transmission to the network after protocol processing by the protocol stack of the host, the message is passed to driver **300**. The INIC then acquires network-sized portions of the message data for that transmission via INIC DMA units, prepends headers to the data portions and sends the resulting message packets down the wire. Conversely, in receiving a TCP, TTCP, SPX or similar message packet from the network **210** to be used in setting up a fast-path connection, miniport driver **306** diverts that message packet to command driver **300** for processing. The driver **300** processes the message packet to create a context for that message, with the driver **302** passing the context and command instructions back to the WIC **200** as a CCB for sending data of subsequent messages for the same connection along a fast-path. Hundreds of TCP, TTCP, SPX or similar CCB connections may be held indefinitely by the INIC, although a least recently used (LRU) algorithm is employed for the case when the INIC cache is full. The driver **300** can also create a connection context for a TTCP request which is passed to the INIC **200** as a CCB, allowing fast-path transmission of a TTCP reply to the request. A message having a protocol that is not accelerated can be processed conventionally by protocol stack **310**.

FIG. 11 shows a TCP/IP implementation of command driver software for Microsoft® protocol messages. A conventional host protocol stack **350** includes MAC layer **353**, IP layer **355** and TCP layer **358**. A command driver **360** works in concert with the host stack **350** to process network messages. The command driver **360** includes a MAC layer **363**, an IP layer **366** and an Alacritech TCP (ATCP) layer **373**. The conventional stack **350** and command driver **360** share a network driver interface specification (NDIS) layer **375**, which interacts with the INIC miniport driver **306**. The INIC miniport driver **306** sorts receive indications for processing by either the conventional host stack **350** or the ATCP driver **360**. A TDI filter driver and upper layer interface **380** similarly determines whether messages sent from a TDI user **382** to the network are diverted to the command driver and perhaps to the fast-path of the INIC, or processed by the host stack.

FIG. 12 depicts a typical SMB exchange between a client **190** and server **290**, both of which have communication devices of the present invention, the communication devices each holding a CCB defining their connection for fast-path movement of data. The client **190** includes INIC **150**, **802.3** compliant data link layer **160**, IP layer **162**, TCP layer **164**, NetBios layer **166**, and SMB layer **168**. The client has a slow-path **157** and fast-path **159** for communication processing. Similarly, the server **290** includes INIC **200**, **802.3** compliant data link layer **212**, IP layer **215**, TCP layer **217**, NetBios layer **220**, and SMB **222**. The server is connected to network lines **240**, **242** and **244**, as well as line **210** which is

15

connected to client **190**. The server also has a slow-path **231** and fast-path **237** for communication processing.

Assuming that the client **190** wishes to read a 100 KB file on the server **290**, the client may begin by sending a Read Block Raw (RBR) SMB command across network **210** requesting the first 64 KB of that file on the server **290**. The RBR command may be only 76 bytes, for example, so the INIC **200** on the server will recognize the message type (SMB) and relatively small message size, and send the 76 bytes directly via the fast-path to NetBios of the server. NetBios will give the data to SMB, which processes the Read request and fetches the 64 KB of data into server data buffers. SMB then calls NetBios to send the data, and NetBios outputs the data for the client. In a conventional host, NetBios would call TCP output and pass 64 KB to TCP, which would divide the data into 1460 byte segments and output each segment via IP and eventually MAC (slow-path **231**). In the present case, the 64 KB data goes to the ATCP driver along with an indication regarding the client-server SMB connection, which denotes a CCB held by the INIC. The INIC **200** then proceeds to DMA 1460 byte segments from the host buffers, add the appropriate headers for TCP, IP and MAC at one time, and send the completed packets on the network **210** (fast-path **237**). The INIC **200** will repeat this until the whole 64 KB transfer has been sent. Usually after receiving acknowledgement from the client that the 64 KB has been received, the INIC will then send the remaining 36 KB also by the fast-path **237**.

With INIC **150** operating on the client **190** when this reply arrives, the INIC **150** recognizes from the first frame received that this connection is receiving fast-path **159** processing (TCP/IP, NetBios, matching a CCB), and the ATCP may use this first frame to acquire buffer space for the message. This latter case is done by passing the first 128 bytes of the NetBios portion of the frame via the ATCP fast-path directly to the host NetBios; that will give NetBios/SMB all of the frame's headers. NetBios/SMB will analyze these headers, realize by matching with a request ID that this is a reply to the original RawRead connection, and give the ATCP a 64K list of buffers into which to place the data. At this stage only one frame has arrived, although more may arrive while this processing is occurring. As soon as the client buffer list is given to the ATCP, it passes that transfer information to the INIC **150**, and the INIC **150** starts DMAing any frame data that has accumulated into those buffers.

FIG. 13 provides a simplified diagram of the INIC **200**, which combines the functions of a network interface controller and a protocol processor in a single ASIC chip **400**. The INIC **200** in this embodiment offers a full-duplex, four channel, 10/100-Megabit per second (Mbps) intelligent network interface controller that is designed for high speed protocol processing for server applications. Although designed specifically for server applications, the INIC **200** can be connected to personal computers, workstations, routers or other hosts anywhere that TCP/IP, TTCP/IP or SPX/IPX protocols are being utilized.

The INIC **200** is connected with four network lines **210**, **240**, **242** and **244**, which may transport data along a number of different conduits, such as twisted pair, coaxial cable or optical fiber, each of the connections providing a media independent interface (MID). The lines preferably are 802.3 compliant and in connection with the INIC constitute four complete Ethernet nodes, the INIC supporting 10Base-T, 10Base-T2, 100Base-TX, 100Base-FX and 100Base-T4 as well as future interface standards. Physical layer identification and initialization is accomplished through host driver initialization routines. The connection between the network lines **210**,

16

240, **242** and **244** and the INIC **200** is controlled by MAC units MAC-A **402**, MAC-B **404**, MAC-C **406** and MAC-D **408** which contain logic circuits for performing the basic functions of the MAC sublayer, essentially controlling when the INIC accesses the network lines **210**, **240**, **242** and **244**. The MAC units **402-408** may act in promiscuous, multicast or unicast modes, allowing the INIC to function as a network monitor, receive broadcast and multicast packets and implement multiple MAC addresses for each node. The MAC units **402-408** also provide statistical information that can be used for simple network management protocol (SNMP).

The MAC units **402**, **404**, **406** and **408** are each connected to a transmit and receive sequencer, XMT & RCV-A **418**, XMT & RCV-B **420**, XMT & RCV-C **422** and XMT & RCV-D **424**, by wires **410**, **412**, **414** and **416**, respectively. Each of the transmit and receive sequencers can perform several protocol processing steps on the fly as message frames pass through that sequencer. In combination with the MAC units, the transmit and receive sequencers **418-422** can compile the packet status for the data link, network, transport, session and, if appropriate, presentation and application layer protocols in hardware, greatly reducing the time for such protocol processing compared to conventional sequential software engines. The transmit and receive sequencers **410-414** are connected, by lines **426**, **428**, **430** and **432** to an SRAM and DMA controller **444**, which includes DMA controllers **438** and SRAM controller **442**. Static random access memory (SRAM) buffers **440** are coupled with SRAM controller **442** by line **441**. The SRAM and DMA controllers **444** interact across line **446** with external memory control **450** to send and receive frames via external memory bus **455** to and from dynamic random access memory (DRAM) buffers **460**, which is located adjacent to the IC chip **400**. The DRAM buffers **460** may be configured as 4 MB, 8 MB, 16 MB or 32 MB, and may optionally be disposed on the chip. The SRAM and DMA controllers **444** are connected via line **464** to a PCI Bus Interface Unit (BIU) **468**, which manages the interface between the INIC **200** and the PCI interface bus **257**. The 64-bit, multiplexed BIU **468** provides a direct interface to the PCI bus **257** for both slave and master functions. The INIC **200** is capable of operating in either a 64-bit or 32-bit PCI environment, while supporting 64-bit addressing in either configuration.

A microprocessor **470** is connected by line **472** to the SRAM and DMA controllers **444**, and connected via line **475** to the PCI BIU **468**. Microprocessor **470** instructions and register files reside in an on chip control store **480**, which includes a writable on-chip control store (WCS) of SRAM and a read only memory (ROM), and is connected to the microprocessor by line **477**. The microprocessor **470** offers a programmable state machine which is capable of processing incoming frames, processing host commands, directing network traffic and directing PCI bus traffic. Three processors are implemented using shared hardware in a three level pipelined architecture that launches and completes a single instruction for every clock cycle. A receive processor **482** is dedicated to receiving communications while a transmit processor **484** is dedicated to transmitting communications in order to facilitate full duplex communication, while a utility processor **486** offers various functions including overseeing and controlling PCI register access. The instructions for the three processors **482**, **484** and **486** reside in the on-chip control-store **480**.

The INIC **200** in this embodiment can support up to 256 CCBs which are maintained in a table in the DRAM **460**. There is also, however, a CCB index in hash order in the SRAM **440** to save sequential searching. Once a hash has

17

been generated, the CCB is cached in SRAM, with up to sixteen cached CCBs in SRAM in this example. These cache locations are shared between the transmit **484** and receive **486** processors so that the processor with the heavier load is able to use more cache buffers. There are also eight header buffers and eight command buffers to be shared between the sequencers. A given header or command buffer is not statically linked to a specific CCB buffer, as the link is dynamic on a per-frame basis.

FIG. **14** shows an overview of the pipelined microprocessor **470**, in which instructions for the receive, transmit and utility processors are executed in three distinct phases according to Clock increments I, II and III, the phases corresponding to each of the pipeline stages. Each phase is responsible for different functions, and each of the three processors occupies a different phase during each Clock increment. Each processor usually operates upon a different instruction stream from the control store **480**, and each carries its own program counter and status through each of the phases.

In general, a first instruction phase **500** of the pipelined microprocessors completes an instruction and stores the result in a destination operand, fetches the next instruction, and stores that next instruction in an instruction register. A first register set **490** provides a number of registers including the instruction register, and a set of controls **492** for first register set provides the controls for storage to the first register set **490**. Some items pass through the first phase without modification by the controls **492**, and instead are simply copied into the first register set **490** or a RAM file register **533**. A second instruction phase **560** has an instruction decoder and operand multiplexer **498** that generally decodes the instruction that was stored in the instruction register of the first register set **490** and gathers any operands which have been generated, which are then stored in a decode register of a second register set **496**. The first register set **490**, second register set **496** and a third register set **501**, which is employed in a third instruction phase **600**, include many of the same registers, as will be seen in the more detailed views of FIGS. **15 A-C**. The instruction decoder and operand multiplexer **498** can read from two address and data ports of the RAM file register **533**, which operates in both the first phase **500** and second phase **560**. A third phase **600** of the processor **470** has an arithmetic logic unit (ALU) **602** which generally performs any ALU operations on the operands from the second register set, storing the results in a results register included in the third register set **501**. A stack exchange **608** can reorder register stacks, and a queue manager **503** can arrange queues for the processor **470**, the results of which are stored in the third register set.

The instructions continue with the first phase then following the third phase, as depicted by a circular pipeline **505**. Note that various functions have been distributed across the three phases of the instruction execution in order to minimize the combinatorial delays within any given phase. With a frequency in this embodiment of 66 Megahertz, each Clock increment takes 15 nanoseconds to complete, for a total of 45 nanoseconds to complete one instruction for each of the three processors. The instruction phases are depicted in more detail in FIGS. **15A-C**, in which each phase is shown in a different figure.

More particularly, FIG. **15A** shows some specific hardware functions of the first phase **500**, which generally includes the first register set **490** and related controls **492**. The controls for the first register set **492** includes an SRAM control **502**, which is a logical control for loading address and write data into SRAM address and data registers **520**. Thus the output of the ALU **602** from the third phase **600** may be placed by

18

SRAM control **502** into an address register or data register of SRAM address and data registers **520**. A load control **504** similarly provides controls for writing a context for a file to file context register **522**, and another load control **506** provides controls for storing a variety of miscellaneous data to flip-flop registers **525**. ALU condition codes, such as whether a carried bit is set, get clocked into ALU condition codes register **528** without an operation performed in the first phase **500**. Flag decodes **508** can perform various functions, such as setting locks, that get stored in flag registers **530**.

The RAM file register **533** has a single write port for addresses and data and two read ports for addresses and data, so that more than one register can be read from at one time. As noted above, the RAM file register **533** essentially straddles the first and second phases, as it is written in the first phase **500** and read from in the second phase **560**. A control store instruction **510** allows the reprogramming of the processors due to new data in from the control store **480**, not shown in this figure, the instructions stored in an instruction register **535**. The address for this is generated in a fetch control register **511**, which determines which address to fetch, the address stored in fetch address register **538**. Load control **515** provides instructions for a program counter **540**, which operates much like the fetch address for the control store. A last-in first-out stack **544** of three registers is copied to the first register set without undergoing other operations in this phase. Finally, a load control **517** for a debug address **548** is optionally included, which allows correction of errors that may occur.

FIG. **15B** depicts the second microprocessor phase **560**, which includes reading addresses and data out of the RAM file register **533**. A scratch SRAM **565** is written from SRAM address and data register **520** of the first register set, which includes a register that passes through the first two phases to be incremented in the third. The scratch SRAM **565** is read by the instruction decoder and operand multiplexer **498**, as are most of the registers from the first register set, with the exception of the stack **544**, debug address **548** and SRAM address and data register mentioned above. The instruction decoder and operand multiplexer **498** looks at the various registers of set **490** and SRAM **565**, decodes the instructions and gathers the operands for operation in the next phase, in particular determining the operands to provide to the ALU **602** below. The outcome of the instruction decoder and operand multiplexer **498** is stored to a number of registers in the second register set **496**, including ALU operands **579** and **582**, ALU condition code register **580**, and a queue channel and command **587** register, which in this embodiment can control thirty-two queues. Several of the registers in set **496** are loaded fairly directly from the instruction register **535** above without substantial decoding by the decoder **498**, including a program control **590**, a literal field **589**, a test select **584** and a flag select **585**. Other registers such as the file context **522** of the first phase **500** are always stored in a file context **577** of the second phase **560**, but may also be treated as an operand that is gathered by the multiplexer **572**. The stack registers **544** are simply copied in stack register **594**. The program counter **540** is incremented **568** in this phase and stored in register **592**. Also incremented **570** is the optional debug address **548**, and a load control **575** may be fed from the pipeline **505** at this point in order to allow error control in each phase, the result stored in debug address **598**.

FIG. **15C** depicts the third microprocessor phase **600**, which includes ALU and queue operations. The ALU **602** includes an adder, priority encoders and other standard logic functions. Results of the ALU are stored in registers ALU output **618**, ALU condition codes **620** and destination oper-

19

and results 622. A file context register 616, flag select register 626 and literal field register 630 are simply copied from the previous phase 560. A test multiplexer 604 is provided to determine whether a conditional jump results in a jump, with the results stored in a test results register 624. The test multiplexer 604 may instead be performed in the first phase 500 along with similar decisions such as fetch control 511. A stack exchange 608 shifts a stack up or down by fetching a program counter from stack 594 or putting a program counter onto that stack, results of which are stored in program control 634, program counter 638 and stack 640 registers. The SRAM address may optionally be incremented in this phase 600. Another load control 610 for another debug address 642 may be forced from the pipeline 505 at this point in order to allow error control in this phase also. A queue RAM and queue ALU 606 reads from the queue channel and command register 587, stores in SRAM and rearranges queues, adding or removing data and pointers as needed to manage the queues of data, sending results to the test multiplexer 604 and a queue flags and queue address register 628. Thus the queue RAM and ALU 606 assumes the duties of managing queues for the three processors, a task conventionally performed sequentially by software on a CPU, the queue manager 606 instead providing accelerated and substantially parallel hardware queuing.

The above-described system for protocol processing of data communication results in dramatic reductions in the time required for processing large, connection-based messages. Protocol processing speed is tremendously accelerated by specially designed protocol processing hardware as compared with a general purpose CPU running conventional protocol software, and interrupts to the host CPU are also substantially reduced. These advantages can be provided to an existing host by addition of an intelligent network interface card (INIC), or the protocol processing hardware may be integrated with the CPU. In either case, the protocol processing hardware and CPU intelligently decide which device processes a given message, and can change the allocation of that processing based upon conditions of the message.

The invention claimed is:

1. A method for network communication by a host computer having a network interface that is connected to the host by an input/output bus, the method comprising:
 - running, on the host computer, a protocol processing stack including an Internet Protocol (IP) layer and a Transmission Control Protocol (TCP) layer, with an application layer running above the TCP layer;
 - initializing, by the host computer, a TCP connection that is defined by source and destination IP addresses and source and destination TCP ports;
 - receiving, by the network interface, first and second packets, wherein the first packet has a first TCP header and contains first payload data for the application, and the second packet has a second TCP header and contains second payload data for the application;
 - checking, by the network interface, whether the packets have certain exception conditions, including checking whether the packets are IP fragmented, checking whether the packets have a FIN flag set, and checking whether the packets are out of order;
 - if the first packet has any of the exception conditions, then protocol processing the first TCP header by the protocol processing stack;
 - if the second packet has any of the exception conditions, then protocol processing the second TCP header by the protocol processing stack;

20

if the packets do not have any of the exception conditions, then bypassing host protocol processing of the TCP headers and storing the first payload data and the second payload data together in a buffer of the host computer, such that the payload data is stored in the buffer in order and without any TCP header stored between the first payload data and the second payload data.

2. The method of claim 1, including allocating the buffer by the application, prior to storing the first payload data and the second payload data together in the buffer.

3. The method of claim 1, wherein storing the first payload data and the second payload data together in a buffer of the host computer is performed by a direct memory access (DMA) unit of the network interface.

4. The method of claim 1, including accumulating the payload data in a memory on the network interface, such that the payload data is stored in the memory in order and without any TCP header stored between the first payload data and the second payload data, prior to storing the first payload data and the second payload data together in the buffer.

5. The method of claim 1, including sending an identification of the TCP connection from the host computer to the network interface, the identification including the source and destination IP addresses and source and destination TCP ports that define the TCP connection, prior to checking whether the packets have certain exception conditions.

6. The method of claim 1, including comparing, by the network interface, the IP addresses and TCP ports of the packets with the source and destination IP addresses and source and destination TCP ports that define the TCP connection.

7. The method of claim 1, wherein checking whether the packets have certain exception conditions includes checking whether the packets have a RST flag set.

8. The method of claim 1, wherein checking whether the packets have certain exception conditions includes checking whether the packets have a SYN flag set.

9. A method for network communication by a host computer having a network interface that is connected to the host by an input/output bus, the method comprising:

- receiving, by the network interface, a first packet having a header including source and destination Internet Protocol (IP) addresses and source and destination Transmission Control Protocol (TCP) ports;

- protocol processing, by the host computer, the first packet, thereby initializing a TCP connection that is defined by the source and destination IP addresses and source and destination TCP ports;

- receiving, by the network interface, a second packet having a second header and payload data, wherein the second header has IP addresses and TCP ports that match the IP addresses and TCP ports of the TCP connection;

- receiving, by the network interface, a third packet having a third header and additional payload data, wherein the third header has IP addresses and TCP ports that match the IP addresses and TCP ports of the TCP connection;

- checking, by the network interface, whether the second and third packets have certain exception conditions, including checking whether the packets are IP fragmented, checking whether the packets have a FIN flag set, and checking whether the packets are out of order;

- if the second packet has any of the exception conditions, then protocol processing the second packet by the host computer;

- if the third packet has any of the exception conditions, then protocol processing the third packet by the host computer;

21

if the second and third packets do not have any of the exception conditions, then storing the payload data of the second and third packets together in a buffer of the host computer, such that the payload data is stored in the buffer in order and without any TCP header stored between the payload data of the second and third packets.

10. The method of claim 9, including allocating the buffer by an application running on the host computer, prior to storing the payload data of the second and third packets together in the buffer.

11. The method of claim 9, wherein storing the payload data of the second and third packets together in a buffer of the host computer is performed by a direct memory access (DMA) unit of the network interface.

12. The method of claim 9, including accumulating the payload data of the second and third packets in a memory on the network interface, such that the payload data of the second and third packets is stored in the memory in order and without any TCP header stored between the payload data of the second and third packets, prior to storing the payload data of the second and third packets together in the buffer.

13. The method of claim 9, including sending an identification of the TCP connection from the host computer to the network interface, the identification including the source and destination IP addresses and source and destination TCP ports that define the TCP connection, prior to checking whether the second and third packets have certain exception conditions.

14. The method of claim 9, including comparing, by the network interface, the IP addresses and TCP ports of the second and third packets with the source and destination IP addresses and source and destination TCP ports that define the TCP connection.

15. The method of claim 9, wherein checking whether the second and third packets have certain exception conditions includes checking whether the packets have a RST flag set.

16. The method of claim 9, wherein checking whether the second and third packets have certain exception conditions includes checking whether the packets have a SYN flag set.

22

17. An apparatus for network communication, the apparatus comprising:

a host computer running a protocol stack including an Internet Protocol (IP) layer and a Transmission Control Protocol (TCP) layer, the protocol stack adapted to establish a TCP connection for an application layer running above the TCP layer, the TCP connection being defined by source and destination IP addresses and source and destination TCP ports;

a network interface that is connected to the host computer by an input/output bus, the network interface adapted to parse the headers of received packets to determine whether the headers have the IP addresses and TCP ports that define the TCP connection and to check whether the packets have certain exception conditions, including whether the packets are IP fragmented, have a FIN flag set, or are out of order, the network interface having logic that directs any of the received packets that have the exception conditions to the protocol stack for processing, and directs the received packets that do not have any of the exception conditions to have their headers removed and their payload data stored together in a buffer of the host computer, such that the payload data is stored in the buffer in order and without any TCP header stored between the payload data that came from different packets of the received packets.

18. The apparatus of claim 17, wherein the buffer is controlled by the application.

19. The apparatus of claim 17, wherein the network interface includes a direct memory access (DMA) unit that is adapted to store the payload data in the buffer.

20. The apparatus of claim 17, wherein the network interface includes a memory that is adapted to accumulate the payload data without any TCP header stored between the payload data that came from different packets of the received packets.

21. The apparatus of claim 17, wherein the exception conditions include having a RST flag set.

22. The apparatus of claim 17, wherein the exception conditions include having a SYN flag set.

* * * * *