Annex AppleTalk Overview

UNIX host

Daemons:
— erpcd – Annex boot service, etc.
— agated?
— CAP/other AppleTalk file/print server
Configuration:
— na – Annex ROM conf utility TBD: AT conf
— acp – Access Control Protocol (will PPP support interface to acp?) TBD: AT net #, zone
Utilities:
— acda – Annex Core Dump Analyzer
— TBD: SUN/376 cross-compiler

LaserWriter

AppleTalk Phase II Seed Router

TBD: Macintosh server?

AppleTalk socket? MacTCP?
- essential erpcd services

Configuration:
- na (acp configuration)

Console: printf debugging

ETHERNET

ANNEX-III

TBD: **AppleTalk Phase II routing**

Phase I: AT-GW MOS-compatible port; AppleTalkARP, protosw entry points for routing, but no socket support for user layers–is it needed later? Liason compatible SLAT?
Phase II: BSD optimization; mbufs adaptation (2 fewer copies), acp support for dynamic AT net # negotiation
Phase III: based on new router code (earlier? not at all?); inter-router forwarding protocol for "secure" source net spoofing.

PPP

TBD: AppleTalk:
AARP/Net-Node Nego

Telnet-3270?

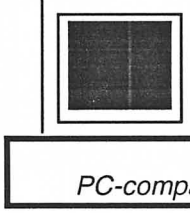
TBD: Telnet Terminal Type nego.
TBD: VT3270:
3278-2 -> VT100 screen mapping
VT100 -> 3270 key mapping

Mac

Mac

Mac File-server

Remote Macintosh/ Router



PC-compatible

PPP Packet Driver? ODI?
TN support? ???

VT100 - ANSI X3.64 compatible terminal

TBD: PPP —> AppleTalk LAP (Network Control Panel adev code resource)
PPP —> vJ-TCP —> MacTCP Link Layer; Comm Toolbox support?
A shared PPP interface with same user interface for all interfaces is desirable.
SLAT: Liasion or Apple Remote Link Protocol interface? ???
Routing support: Mac+ bridge? Liasion? Apple? Req'd for convenient remote ATalk Laser printing.

Community Access Project

Network Resources/Technologies & Standards

Project Summary:

The goal of the Community Access Project is to allow remote access via telephone to networks connected to the Cornell backbone which employ the AppleTalk and TCP/IP protocol stacks. Recent reductions in the cost of V.32 standard 9600-baud modems make it possible to provide an economical high speed dial-up service which can run overhead-laden network protocols with acceptable performance. Combined with recent advances in modem-based data compression and TCP/IP header compression, V.32 technology allows high-performance terminal emulation, downloading, and mail service, and low-performance file-sharing to be extended beyond the reach of the Cornell campus wire plant.

To accomplish this goal, we propose to enhance Cornell's PC TN, PC C-Gateway, and Macintosh OmniTalk/Bridge to support modem links, and to develop a SLIP (Serial Line Internet Protocol) driver for Apple's MacTCP. Building on the C-Gateway platform to develop the CAP-gateway will allow us to provide security and accounting facilities which will allow CIT to levy charges on users of these telecommunication services.

Project Phases:

Phase I Serial Line Internet Protocol (RFC 1055) support.

Products Basic SLIP support to enable fruitful access to hosts on the Cornell network.

- A: Test existing SLIPs: CMU PC-IP Telnet, KA9Q IP router (1/1/90).
- B: Implement SLIP driver for MacTCP (MacSLIP, 1/30/90).
- C: Implement SLIP/SLAP driver for C-Gateway (CAP-Gateway, 3/30/90).
- D: Implement SLAP driver for Cornell OmniTalk/MacBridge (MacSLAP,

Phase II SLIP header compression.

Product Compressed SLIP support to improve performance with ASCII hosts.

- A: Implement for MacTCP and CAP-Gateway (4/31/90).

Phase III Expanded services.

- A: Test Point-to-Point Protocol (draft RFC) implementations (KA9Q).
- B: Choose PPP or alternative algorithm for supporting multiple protocols.
- C: Implement multiple protocol support for MacTCP and CAP-Gateway (5/31/90).
- C: Design Access Control and Accounting Interface (2 weeks).
- D: Implement Access Control & Accounting on CAP-Gateway and Community Access Control Server (.

Summary of Direct Costs:

Software Development System (DOS under UNIX)		5,350.00
Development Testbeds and Hardware		9,600.00
Modems		6,470.00
	Total:	$21,420.00

Itemized Budget for Direct Costs

The direct costs of equipment, software, and services required to conduct this project are estimated as follows:

Software Development System (DOS under UNIX):

	Informtech 386 (25MHz)	1		1,200.00
	4MB memory	4	250.00	1,000.00
	Imprimis 320MB disk	1		2,100.00
	WD Ethernet card	1		300.00
	VGA graphics adapter	1		100.00
	monochrome monitor	1		150.00
	SCO Open Desktop	1		500.00
	(includes UNIX SVr3, DOS, TCP/IP, X-windows)			
			Workbench total:	5,350.00

Development Testbeds and Hardware

	Informtech 286	2	800.00	1,600.00
	(1 for C-Gateway, 1 for PC Telnet)			
	Pronet adapter	1		500.00
	Serial I/O adapters	4	1,000.00	4,000.00
	Serial data scope	1		3,000.00
	analog phone lines	2	250.00	500.00
			Testbed total:	9,600.00

Modems

	V.32/MNP5 modems (Prometheus)	4	650.00	2,600.00
	V.32/MNP5 modems (Racal-Vadic)	2	835.00	1,670.00
	V.32/MNP5 modems (Racal-Vadic)	2	1100.00	2,200.00
(opt)	V.32/MNP9 modems (Microcom)	2	1100.00	
(opt)	V.32/PEP modems (Telebit)	2	900.00	
			Modem total:	6,470.00
			Grand Total	$21,420.00

Description of Project Phases

Phase I: Serial Line Internet Protocol (RFC 1055) support.

Products: Basic SLIP/SLAP support to enable high-performance access to hosts on the Cornell network: Telnet on the Mac & PC, AppleTalk on the Mac.

A: Test existing SLIPs: UNIX, CMU PC-IP Telnet, KA9Q IP router (1/1/90).

Several implementations of SLIP on the PC are freely available for non-commercial use. To ensure compatibility of the CAP server, we need to configure some of these platforms for testing.

B: Implement SLIP driver for MacTCP (1/30/90).

The MacTCP Control Panel device allows the user to reconfigure to use alternate network drivers. Apple MacTCP developer John Veizades has agreed to share the interface specification so that we can develop a SLIP driver for MacTCP. This will allow multiple TCP/IP services on a single remote workstation using a manually assigned address.

C: Implement SLIP/SLAP driver for MTEX/C-Gateway (3/30/90).

The Cornell C-Gateway has been developed internally as a secure, low-cost, median performance router for IP and AppleTalk. Configured with intelligent serial port boards, the C-Gateway could easily handle 16-32 modem connections at a time. This driver will allow the C-Gateway to provide basic SLIP or SLAP connectivity to the Cornell network.

D: Implement SLAP driver for Cornell OmniTalk/MacBridge

Modify the OmniTalk Bridge to function as an AppleTalk SLAP driver. (requires implementation of AARP address acqusition if multiple nodes/net on gateway.)

Phase II: SLIP header compression (forthcoming Van Jacobson RFC).

Product: Compressed SLIP support to improve performance with ASCII hosts.

A: Implement for MacTCP and CAP-Gateway (4/31/90).

Phase III: Expanded services.

A: Test Point-to-Point Protocol (draft RFC) implementations (KA9Q).

C: Design Access Control and Accounting Interface (2 weeks).

D: Implement Access Control & Accounting on CAP-Gateway & Community Access Control Server.

Issues Associated with Dialup Network Service

Address assignment and security:

The remote user must be interactively assigned a network address in order to avoid address collisions.

The remote user may want the option of negotiating a high-security AppleTalk network address through a service such as Kerberos. The gateway would draw on a database server on the backbone to verify the user's identity and get the net number the user has requested. N.B.: This service would consume AppleTalk net space rapidly.

Performance:

The current generation of modem technology is typified by 9600-baud full-duplex V.32 modem, with optional protocols at the link layer which work either to enhance reliability by providing error detection and retransmission of suspect data (MNP1-4, CCITT V.42) or to speed transmission by compressing the data.

Data compression (e.g. MNP5, V.42bis (?)) performed by the modem degrades response time for small network packets, while dramatically improving throughput for larger volumes of data which are comprised of ASCII text.

Interactive terminal services require fast response times. ASCII terminal performance in particular suffers badly due to network protocol overhead and compression misfires. IBM 3270 terminals should perform better, since they provide for local editing of data which is transmitted only at the user's request and only when it has been modified. In order to improve the interactive performance of TCP/IP, Van Jacobsen and others have done research on reducing the size of the TCP/IP header from 40 bytes down to about 6-8 bytes; an RFC (Request for Comments, as Internet protocol specifications are titled) is somewhat over-due from Van Jacobsen on his compression algorithms.

Reliability:

Integrity of transmitted data. TCP/IP provides a ones-complement checksum for a minimal test of end-to-end data integrity; MNP1-4 or LAP-D ensure modem link layer data integrity. AppleTalk may also use a 16-bit ones complement checksum.

Gateway reliability.

Line reliability. The quality of local telephone connections will be poor for some time to come.

Upgradability:

The platform should support higher-speed serial interfaces, e.g. 56Kb/s, which may be supported in the envisionable future by ISDN switches, and 38.4Kb/s, which is now supported by high-speed modems from Microcom and USR (the $1100 Microcom QX/V.32c modem with MNP1-9 reportedly achieves throughput as high as 3100 cps for ASCII text; the CCITT V.42bis specification, which several modem vendors will soon implement, uses the Lempel-Ziv compression algorithm and should provide yet higher throughput).

Serial Line Link Layer Protocols

SLIP (Serial Line Internet Protocol)

A very simple protocol, which frames a packet and does nothing else.

Drawbacks:

Fixed IP address. No standard scheme for negotiating IP addresses.
Link transmission errors must be detected and corrected at TCP or other higher protocol level.

SLAP (Serial Line AppleTalk Protocol)

Relatively simple; the framing is the same as SLIP.
AppleTalk services work transparently with willing AppleTalk networks.
TCP/IP address assignment will work automatically as it does on an AppleTalk.

Drawbacks:

Error recovery, as with SLIP.
AppleTalk packet overhead: AppleTalk packets use a 8-byte header for its LAP (link layer, 3 bytes) and short DDP (network layer, 5 bytes) encapsulation for AppleTalk packet which destined for a node on the local network (in this case the CAP-Gateway), which would be added to the 40-byte minimum length of a TCP packet if one uses TCP/IP services over AppleTalk.
TCP/IP header compression may not achieve its goal of reducing the a packet with a single byte of data (the typical ASCII terminal interaction) size below the threshold at which modem compression schemes begin to add overhead, as they attempt to compress packet headers--which do not compress well at all.

PPP (The Point-to-Point Protocol)

An Internet RFC will be released for PPP, which is intended to provide a standard for transmitting multiple protocols over dial-up connections.
A flexible option structure is provided. Options are included in the draft to negotiate IP addresses and select data compression techniques.
PPP offers negotiated character mapping to appease picky links.
Several vendors intend to provide PPP on dialup platforms, and UCDavis is developing PPP functionality on the KA9Q Telnet/Router base, which already supports SLIP.

Drawbacks:

PPP packet overhead: PPP specifies the use of synchronous framing based on HDLC on asynchronous links, which adds an 8-byte header with a 16-bit CRC checksum. You can negotatiate a minimum header size of 5 bytes sans CRC. As with AppleTalk, this may impair TCP/IP header compression performance.
Complexity: PPP is substantially more complex than SLIP/SLAP, since it strives to provide general solutions for a broad base of applications (e.g. connections over X.25 links).

NOTES ON THE OMNITALK WORKSTATION/BRIDGE CODE

There are a number of factors which have affected the final form of the system.

PURE WORKSTATION CASE:

The Omninet driver for the Macintosh does not perform double-buffering, so the AppleTalk LAP code must return ASAP to the Omninet driver so it can get another buffer and set up another receive on the Omninet card. In the case where the driver is handling large packets, several retries are common; with bursts of small packets, more than ten retries are often necessary, so the driver sends the packet three times if necessary.

Synchronicity: Unfortunately many calls to the AppleTalk driver are made synchronously, so upcalls to the AppleTalk stack cannot rely on obtaining cycles from a driver (such as the .bridgeDA) running in "normal" time but must instead perform the upcalls synchronously from interrupt level. Due to the problem mentioned above, we therefore use the millisecond timer to trigger the upcall.

Memory: Apple has had several conflicting (and confusing) approaches to allocating memory for drivers, and the technique used by the Omninet driver does not work for an Alternate AppleTalk implementation. At Alternate LAP install time, which occurs early in the Startup process before INITS are called, memory in the System Heap is limited, so the 'CVIT' resource is truncated so the system will fit. Unfortunately this resource includes the timer for OmniDrive services, so the current version does not support OmniDrives.

PURE BRIDGE CASE:

In the bridge case the forwarding of packets is performed during "normal" time by getting cycles from the .bridgeDA. Forwarding of packets at interrupt time leads to failure. Alas, some Mac applications, notably the Finder, are less than generous when providing driver cycles through calls to SystemTask, slowing bridge performance. (The "race" application demonstrates the effect—the MPW Shell also shares cycles properly.)

Memory: The bridge requires numerous buffers, so the following solution was worked out: at Install time, the OmniTalk code defers memory allocation, instead opening the .bridgeDA driver and calling it with pointers to routines to initialize OmniTalk and service its queues. Later, at INIT time, the System calls the 'OmniINIT' INIT, which contains an 'sysz' resource which causes the System to expand the System Heap, and which then calls the .bridgeDA to complete the initialization of OmniTalk.

MIXED WORKSTATION/BRIDGE CASE:

This case adds another complication: NBP calls often need to be performed synchronously, producing two broadcast packets from the bridge in the case where the Zone names are identical, yet the AppleTalk .MPP driver is not re-entrant. This case has been handled for the Mac+ by patching the .MPP variable containing the WDS, but this has not been fixed for the MacSE/II/etc. This requires obtaining the .MPP version # and the corresponding location of the stored WDS for each version.

Components and functions:

.bridgeDA:	a driver which is called to initialize OmniTalk and which calls queue service routines in OmniTalk
!OmniINIT:	an INIT which obtains memory on the System Heap and then calls .bridgeDA to cause the remainder of the OmniTalk installation process to be performed

OmniTalk resource code:

omniadev.a:	interfaces to LAP Mgr. Network Chooser interface so user can select OmniTalk
omniatlk.a:	contains code which sends and receives packets for LAP mgr, along with code which interfaces to the C language bridge code
bridge.c:	code for the bridging functions
bridgeintf.c:	code which interfaces bridge to omniatlk routines

Additional code required to make DA & INIT:

bridgeDA.c:	code for the DA which co-ordinates startup & shares cycles for bridge functions.
bridgeinit.a:	code for the INIT resource which acquires System Heap memory & triggers startup.

Trivial application:

race.c	shares cycles liberally with DA's.

Header files:

lapmgrequ.a	equates needed by LAP Mgr.
omni.equ	equates for omnitalk
bridge.h	bridge declarations

Corvus OmniDriver/PTD resources:

'Corvus Modem Port'	contains modified PTD, MacSpie, & OmniDriver resources
modPTD	.PTD patched to avoid ID conflict with QuickMail
'omnidriver+'	modified OmniDriver (handles completion calls correctly)—source in OmniDriver folder (unchanged from the previous OmniTalk version)
'small macspie'	stub Macspie to fit into MacII--eliminates VBL timer code needed to do ConstellationIII

Makefiles, resource files, etc.

bridgeDA.DRVR	intermediate file
bridgeDA.DRVW	intermediate file
bridgeDA.make	
bridgeDA.r	
bridgeinit.make	
bridgeinit.r	
bridgeinit.rsrc	
bridgeoff.rsrc	bridge configuration resource included in Omnitalk.r to set bridge off
bridgeon.rsrc	bridge configuration resource included in Omnitalk.r to set bridge on
makefile	makefile for OmniTalk
omniatlk.map	map of variables in Omnitalk code
OmniTalk	resource to be placed in System Folder
OmniTalk.r	
omnitalk.rsrc	
race.make	
worksheet	MPW worksheet to compile code (compiled under MPW 2.0.2)

Documents:

'Omninet Installation Notes'	describes installation procedure for OmniTalk
'OmniTalk design desiderata.W'	this document.
'OmniTalk ARP'	describes AppleTalk address resolution under OmniTalk

Assorted bugs:

The Omninet driver should be fixed so that it allocates memory and initializes only on the first open, so OmniTalk does not rely on a fixed ID (31--changed to avoid conflict with the MacroMaker DA).

Three driver slots are used (.bridgeDA, .Omnidriver, .PTD) where one should suffice.

OMNITALK ADDRESS RESOLUTION PROTOCOL

Address mapping from AppleTalk protocol addresses to Omninet hardware addresses

CIT Network Planning and Development

The following document describes the recommended method for acquiring an AppleTalk address dynamically on an Omninet network. Implementors of AppleTalk LAP layers on Omninet should adhere to this standard in order to guarantee compatibility.

At the time a host initializes an Omninet transporter to serve as an AppleTalk node, the host software should first attempt to use the current switch settings on the transporter as its node number. As is now the case, the transporter should use Echo Protocol to ensure the uniqueness of this address. If there is another transporter with the same address, the host software should set the transporter address to another value using the Poke command, and once again attempt to confirm its uniqueness using Echo Protocol. The host software should continue attempting to discover an address until it discovers that all available addresses are in use, in which case it should report the error condition to the user.

When a network node chooses a LocalTalk address, the client software on a Macintosh can indicate to the AppleTalk Manager that it wishes to acquire a "server address" (128-254) rather than a "workstation address" (1-127). The LocalTalk LAP layer then performs a more thorough address-conflict test (sending some 4500 ENQ packets over 8 seconds, rather than some 600 ENQ packets over 1 second).

The Omninet Echo Protocol offers a superior alternative to the use of AARP probe packets (as defined in the Apple EtherTalk and Alternate AppleTalk Reference), since an initialized transporter is almost always prepared to respond to an Echo packet without host intervention. Furthermore, there is no need for AARP-like address resolution on Omninet because Omninet transporters only recognize 64 addresses on a network: since any AppleTalk address (except the broadcast address, 255) can be mapped into the Omninet address space by masking it with $3F, the Omninet local address space is a tidy subset of the AppleTalk local address space.

Therefore, when the client software requests an address, we can use our Omninet hardware address to determine the AppleTalk protocol address. If the client requests a workstation address, we simply use the transporter address. If the client requests a server address, we add 128 to the transporter address to honor that request. Only one node on an OmniTalk network requires two distinct protocol addresses: the Macintosh-based AppleTalk bridge. This bridge should acquire a server address to guarantee address uniqueness on the LocalTalk side; it can choose its bridge protocol address by adding 64 to the server address already selected for the shared protocol address used by the LocalTalk bridge and workstation elements of the system. Note that there does exist one restriction in this scheme: the bridge transporter address cannot be 63, since this would cause a conflict with the AppleTalk broadcast address.

Algorithms:

```
AppleTalk protocol address determination:
        Workstation_address = Omninet_address < 63;
        Server_address      = Omninet_address + 128;
        Bridge_address      = Omninet_address + 192;

Packet forwarding at the LAP layer:
        Omninet_address     = AppleTalk_protocol_address & 0x3f;

Packet reception:
        if (AppleTalk_protocol_address == 255) {
                do_bridge_forward();
                do_AppleTalk_upcall();
        }
        else if (AppleTalk_protocol_address >= 192)
                do_bridge_forward();
        else
                do_AppleTalk_upcall();
```

Notes on OmniTalk, an Alternate AppleTalk implentation on Corvus Omninet . . .

OmniTalk has 3 components:

• OmniTalk: place in System Folder: an 'adev' resource for the 'Network' 'cdev' Control Panel device. The 'BRDG' and 'STR ' resources in this file control configuration of the OmniTalk system. The formats are as follows:

BRDG format:

WORD: Enable bridge operation; if non-zero, the workstation will be configured as a bridge between the attached Omninet and LocalTalk networks. Currently the bridge only runs on a Mac+.
WORD: Omninet AppleTalk net number. Unused if not configured as a bridge.
WORD: LocalTalk AppleTalk net number. Unused if not configured as a bridge.
WORD: Number of buffers to allocate for the OmniTalk system.

STR format:

STR	1	is the name of the LocalTalk Zone;
STR	2	is the name of the OmniTalk Zone;

• !OmniINIT: place in System Folder: an 'INIT' resource which handles memory allocation for the OmniTalk system. By modifying the 'sysz' resource (a long number, 4 bytes) in !OmniINIT one can change the size of the System Heap; if you wish to dramatically increase the number of buffers, you should increase 'sysz'.

• .bridgeDA: install System File using ResEdit: a 'DRVR' resource which gets CPU time from the System so bridge I/O can be performed in normal processing time rather than at interrupt time.

Unfortunately the system currently requires 3 driver slots for its operation, and two of these have numbers which cannot be easily changed. The '.bridgeDA' driver is number 31, but it can be renumbered as required. Two other drivers, the Corvus '.PTD', number 23, and the Corvus 'OmniDriver', number 30, are installed at the time OmniTalk is initialized. If slots 23 and 30 are not open, installation at boot time will fail, resulting in the message "Cannot install Alternate AppleTalk, using built-in AppleTalk instead."