DOCKET FILE COPY ORIGINAL

EX PARTE OR LATE FILED

AT&T

Richard N. Clarke

Room 5462C2
295 North Maple Avenue
Basking Ridge, New Jersey 07920
(908) 221-8685

May 12, 1998

Ms. Magalie Roman Salas
Secretary
Federal Communications Commission
1919 M. St., NW, Room 222
Washington, D.C. 20554

RECEIVED

MAY 12 1998

FEDERAL COMMUNICATIONS COMMISSION
OFFICE OF THE SECRETARY

RE: Ex Parte Presentation – Proxy Cost Models
CC Docket No. 96-45

Dear Ms. Salas:

The attached materials were provided to Bryan Clopton, Chuck Keller, Bob Loube, Jeff Prisbrey and Brad Wimmer of the Common Carrier Bureau.

The first items demonstrate that distances calculated by computing the minimum spanning tree (strand map) between house geocode points typically may exceed the amount of distribution backbone and branch cable required to serve these houses. This is because the actual geocode points in the HAI Model are offset 50 feet from the road centerline – towards the appropriate side of the street as given by the address' house number. Thus, the strand map distance will incorporate not just the distribution backbone and branch distance engineered by the HAI Model, but may also include a substantial portion of the drop cable distances. As the attached spreadsheets show, when these appropriate portions of drop (which is engineered with a separate cable "budget" in the HAI Model) are recognized, it is not true that backbone and branch distance should exceed strand map distance.

The two latter attachments are the C++ source code used by PNR in calculating the strand map distances between geocode points.

Two copies of this Notice are being submitted to the Secretary of the FCC in accordance with Section 1.1206(a)(2) of the Commission's rules.

Sincerely,

Richard N. Clarke /ha

Richard N. Clarke

Attachments

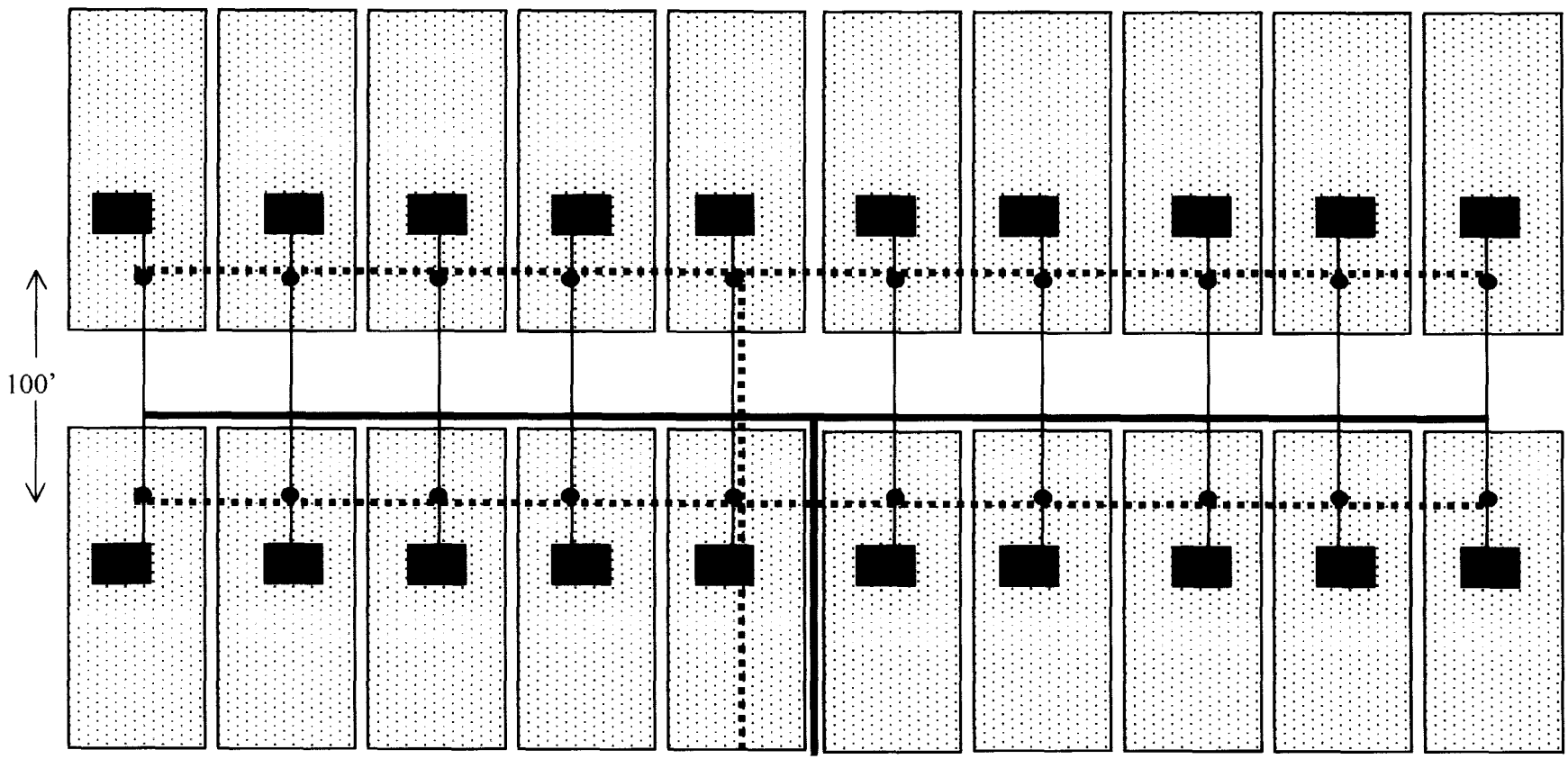
cc: Donald Stockdale
Charles Keller
Bryan Clopton
Bill Sharkey
Brad Wimmer
Robert Loube
Jeff Prisbrey
Sheryl Todd

No. of Copies rec'd 0+1
List ABCDE

Recycled Paper

Distribution Cable Lengths vs. Strand Distances

Next-Door House Geocodes Closer than Across-Street House Geocodes



House

● **Geocode Point** (offset 50' from street centerline)

Rectangular Lots (w wide, 2w deep, w<100')

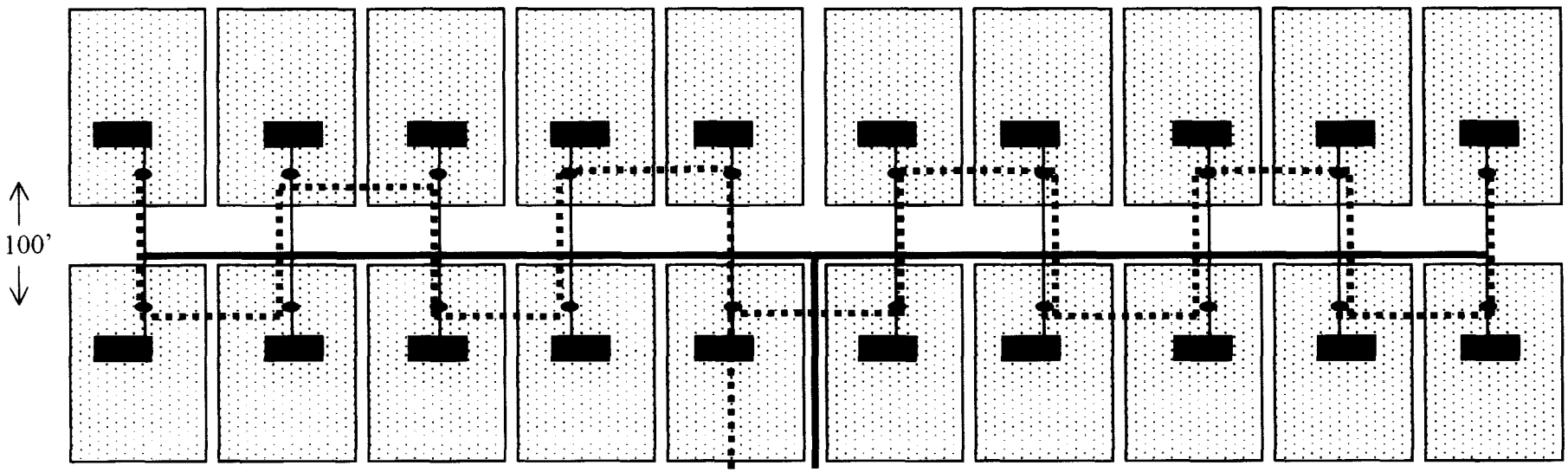
Backbone or Branch Cable

Drop Cable

Strand Map

Distribution Cable Lengths vs. Strand Distances

Across-Street House Geocodes Closer than Next-Door House Geocodes



House

● **Geocode Point** (offset 50' from street centerline)

Rectangular Lots (w wide, 2w deep, w>100')

Backbone or Branch Cable

Drop Cable

Strand Map

Strand Map Distance Will Commonly Exceed the Required Amount of Distribution Route Distance Because it Includes Some Portion of the Drop

Next-door house geocodes closer than across-street house geocodes

Number of lots:	20	
Lot width:	w	w < 100'
Lot depth:	2w	

Strand Map Distance

Vertical:	2w - 50'	Horizontal:	9w
	100'		9w
Subtotal:	2w + 50'	Subtotal:	18w
Total SMD:	20w + 50'		
Amount of drop distance implicitly included in SMD:	20 * 50'		
	1000'		

Required Distribution Route Distance

Backbone:	2w
Branch:	9w
Required DRD:	11w

Compare Strand Map Distance to Required Distribution Route Distance

SMD	>	**DRD**
20w + 50'		11w

Note that if the 1000' of drop distance implicitly included in the SMD is also added to the DRD, this augmented DRD will exceed the SMD because w < 100'

SMD	<	**DRD + allocated drop**
20w + 50'		11w + 1000'

Strand Map Distance Will Commonly Exceed the Required Amount of Distribution Route Distance Because it Includes Some Portion of the Drop

Across-street house geocodes closer than next-door house geocodes

Number of lots:	20	
Lot width:	w	w > 100'
Lot depth:	2w	

Strand Map Distance

Vertical:	2w - 50'	Horizontal:	9w
	10 * 100'		
Subtotal:	2w + 950'	Subtotal:	9w
Total SMD:	11w + 950'		
Amount of drop distance implicitly included in SMD:	20 * 50'		
	1000'		

Required Distribution Route Distance

Backbone:	2w
Branch:	9w
Required DRD:	11w

Compare Strand Map Distance to Required Distribution Route Distance

SMD	>	**DRD**
11w + 950'		11w

Note that if the 1000' of drop distance implicitly included in the SMD is also added to the DRD, this augmented DRD will exceed the SMD because w > 100'

SMD	<	**DRD + allocated drop**
11w + 950'		11w + 1000'

```
// CPath.h : header file

#ifndef PATH__H
#define PATH__H

#include "Status.h"
#include "Distance.h"

#include <stdio.h>
#include <math.h>

/////////////////////////////////////////////////////////////////////////////
// CPath

class CPath : public CObject
{
// Construction
public:
      CPath();     // standard constructor
      virtual ~CPath();

// Attributes
protected:
      int m_nCases;
      double m_Dist;
      double m_DistFlag;
      double m_PointMax;
      CStatus* m_Status;

// Functions
public:
      void Run(CDistance&, CUIntArray&, unsigned int, CUIntArray* Flag = NULL,
CStatus* Stat = NULL);
      void Run(CDistance&, CUIntArray&, CUIntArray&, CUIntArray* Flag = NULL,
CStatus* Stat = NULL);

      int GetNCases();
      double GetDist();
      double GetDistFlag();
      double GetPointMax();

protected:
      void Init(CUIntArray&, CUIntArray&, CUIntArray&, CUIntArray&);
      void Calculate(CUIntArray&, CUIntArray&, CDistance&, CUIntArray* Flag =
NULL);
};

#endif
```

```
// Path.cpp : implementation file

#include "stdafx.h"
#include "Path.h"

CPath::CPath()
{
	m_nCases = 0;
	m_Dist = 0.0;
	m_DistFlag = 0.0;
	m_PointMax = 0.0;
	m_Status = NULL;
}

CPath::~CPath()
{
}

void CPath::Run(CDistance& DistMat, CUIntArray& Path, UINT nCases, CUIntArray*
Flag /*NULL*/, CStatus* Stat /*NULL*/)
{
	// Initialize computed variables
	m_Dist = 0.0;
	m_DistFlag = 0.0;
	m_PointMax = 0.0;

	// Initialize with parameter variables
	m_nCases = nCases;
	m_Status = Stat;

	// Initialize case array for ALL cases
	CUIntArray Cases;
	Cases.SetSize(nCases);
	for (UINT i = 0; i < nCases; i++)
		Cases[i] = i;

	// Calculate path and total distance
	Calculate(Cases, Path, DistMat, Flag);

	return;
}

void CPath::Run(CDistance& DistMat, CUIntArray& Path, CUIntArray& SpecificCases,
CUIntArray* Flag /*NULL*/, CStatus* Stat /*NULL*/)
{
	// Initialize computed variables
	m_Dist = 0.0;
	m_DistFlag = 0.0;
	m_PointMax = 0.0;

	// Initialize with parameter variables
	m_nCases = SpecificCases.GetSize();;
	m_Status = Stat;

	// Calculate path and total distance
	Calculate(SpecificCases, Path, DistMat, Flag);
```

```
	return;
}

////////////////////////////////////////////////////////////////////////////////
////
//
//  CPath:  Path functions
//
////////////////////////////////////////////////////////////////////////////////
////

void CPath::Init(CUIntArray& Cases, CUIntArray& Assigned, CUIntArray&
Unassigned, CUIntArray& Path)
{
	// Initialize path and cases
	Unassigned.SetSize(m_nCases);
	for (int i = 0; i < m_nCases; i++)
	{
		// Initialize all case paths
		Path[Cases[i]] = 0;

		// Add cases to unassigned list
		Unassigned[i] = Cases[i];
	}

	// Set the first assignment
	Path[Cases[0]] = 1;
	Assigned.Add(Cases[0]);
	Unassigned.RemoveAt(0);

	return;
}

void CPath::Calculate(CUIntArray& Cases, CUIntArray& Path, CDistance& DistMat,
CUIntArray* Flag /*NULL*/)
{
	// Declare variables
	double closest, dist;
	unsigned int pt1, pt2, i, j, iLoop, jLoop;
	CUIntArray Unassigned, Assigned;

	// Initialize
	Init(Cases, Assigned, Unassigned, Path);

	// Determine path
	CString origMsg = m_Status->GetText();
	while (Unassigned.GetSize())
	{
		pt1 = 0;
		pt2 = 0;
		closest = DistMat.Get(Assigned[0], Unassigned[0]);

		iLoop = (unsigned)Assigned.GetSize();
		jLoop = (unsigned)Unassigned.GetSize();

		if (m_Status)
			m_Status->SetText(origMsg, iLoop);
```

```
            // Assign household to path
            for (i = 0; i < iLoop; i++)
                  for (j = 0; j < jLoop; j++)
                  {
                        dist = DistMat.Get(Assigned[i], Unassigned[j]);

                        if (dist > m_PointMax)
                              m_PointMax = dist;

                        if (dist < closest)
                        {
                              pt1 = i;
                              pt2 = j;
                              closest = dist;
                        }
                  }

            // Add to path distance
            m_Dist += closest;
            if (Flag && (Flag->GetAt(Assigned[pt1]) == 1 || Flag-
>GetAt(Unassigned[pt2]) == 1))
                  m_DistFlag += closest;

            // Assign and remove
            Path[Unassigned[pt2]] = Path[Assigned[pt1]] + 1;
            Assigned.Add(Unassigned[pt2]);
            Unassigned.RemoveAt(pt2);
            Unassigned.FreeExtra();
      }

      return;
}

/////////////////////////////////////////////////////////////////////////////
////
//
//  CPath:  Attribute functions
//
/////////////////////////////////////////////////////////////////////////////
////

int CPath::GetNCases()
{
      return m_nCases;
}

double CPath::GetDist()
{
      return m_Dist;
}

double CPath::GetDistFlag()
{
      return m_DistFlag;
}
```

```
double CPath::GetPointMax()
{
	return m_PointMax;
}
```