The Application of Data Clustering Algorithms in Packet Pair/Stream Dispersion Probing of Wired and Wired-cum-Wireless Networks

This thesis is submitted in partial fulfilment of the requirements of Kingston University for the Degree of Doctor of Philosophy

by

Mehri Hosseinpour

Faculty of Science, Engineering and Computing

April 2012

ABSTRACT

This thesis reports a study of network probing algorithms to wired and wireless Ethernet networks. It begins with a literature survey of Ethernet and related technology, and existing research on bandwidth probing. The *OPtimized Network Engineering Tool* (OPNET) was used to implement a network probing testbed, through the development of packet pair/stream modules. Its performance was validated using a baseline scenario (two workstations communicating directly on a wired or wireless channel) and it was shown how two different probe packet sizes allowed link parameters (bandwidth and the inter-packet gap) to be obtained from the packet pair measurements and compared with their known values. More tests were carried out using larger networks of nodes carrying cross-traffic, giving rise to multimodal dispersion distributions which could be automatically classified using data-clustering algorithms.

Further studies used the ProbeSim simulation software, which allowed network and data classification processes were brought together in a common simulation framework The probe packet dispersion data were classified dynamically during operation, and a closed-loop algorithm was used to adjust parameters for optimum measurement. The results were accurate for simple wired scenarios, but the technique was shown to be unsuitable for heterogeneous wired-cum-wireless topologies with mixed cross-traffic.

TABLE OF CONTENTS

ACKNOWLEDGMENT

I would like to express my deep and sincere acknowledgment to my supervisor, Dr. Martin Tunnicliffe . His wide knowledge and logical way of thinking have been of great value for me. His understanding, encouraging, personal guidance, supervision and support from the very beginning has enabled me to develop a deep understanding of the subject. Without his help and support, it would have been impossible for me to finish this thesis.

I must express my loving gratitude towards my dearest daughter Yasmin who encouraged and supported me all along the way by being patient and understanding.

I offer my thanks also to Dr. Maria Winnett and other members of staff in Kingston University, who helped me in many ways during the completion of the project. Without their cooperation I could not have obtained much of my relevant study.

I must express my loving gratitude towards my dearest and nearest friend Vida Nezamabad who encouraged me to start the PhD and supported me all along the way by supporting me with her endless love and wisdom every single day in these years and enabled me to accomplish this work. I have to mention a few more friends which always been there for me whenever I needed them including Mrs. Rajshri Shah , Mr. Bijan Jalali and Dr. Mehdi Froughi Fard.

Last but not least I would like to thank my parents Mr. Hossein Hosseinpour and Mrs. Rezvan Khoramshahi, whose endless love and support always encouraged me to be who I am. Also, I could not sustain this work without that crucial love from my brothers and sisters specially Zahra Hosseinpour, the most loving and caring sister in the world.

1. INTRODUCTION

1.1. General Background

There are many reasons why it is useful to measure the bandwidth, latency and other end-to-end properties of paths through a data network. The problem of measuring bandwidth in *wired-cum-wireless* networks is addressed in this thesis.

With the growth of the Internet and the emergence of different wired and wireless technologies, end-to-end paths often combine wired and wireless links. The wired portion is normally at the access level, such as a Wi-Fi home router or a WiMAX network which users may access across a metropolitan-scale region. Wireless networks are becoming increasingly popular and are being installed almost everywhere.

With the rapid expansion of the number of notebook computers and other portable devices, we can anticipate a revolution in wireless mobile Internet similar to that seen with mobile phones. There are basically three types of wireless network: The most common is based on the IEEE 802.11 Wi-Fi standards for a wireless Local Area Network (LAN). These can be both "infrastructure" networks (where end-hosts communicate with a common central router or access-point) and "ad-hoc" networks where the end-hosts themselves act as routers. A second type of wireless network is based on the IEEE 802.15 Bluetooth standards for communication over extremely short distances – typically within a single room or between devices on a desktop. It is cheaper than Wi-Fi and requires less power consumption, but has a significantly smaller throughput. Thirdly there are wireless technologies for communication over wide geographical areas. The principal competitors for this market are 3G (including 3GSM/UMTS, HSDPA, developed for mobile phone technology) and IEEE 802.16 WiMax. WiMax has two main variants: the 802.16d "Fixed WiMax" and the newer 802.16e "Mobile WiMax", neither of which is compatible with

the other. In a wired-cum-wireless environment, these links operate in tandem with wired technologies.

There are still considerable challenges to be addressed in the monitoring of bandwidth in wireless networks. This thesis addresses the end-to-end probing of network paths consisting of wired and wireless nodes, and proposes a new approach based on data clustering algorithms and feedback control.

1.2. What is Bandwidth?

Analogue bandwidth (sometimes called "radio frequency bandwidth", "signal bandwidth", "frequency bandwidth" or "spectral bandwidth"(Glover and Grant 2004) is the difference between upper and lower cutoff frequencies (Hz) for a communication channel. For a particular level of noise, bandwidth determines the rate R at which information can be sent, via the Shannon-Hartley Theorem

$$R = B \cdot \log_2(1 + S/N) \text{ bits per second} \tag{1.1}$$

where B is the analogue bandwidth and S/N is the signal-to-noise ratio. Since R is proportional to B, the word "bandwidth" is also used to denote the channel capacity in bits per second. To avoid ambiguity we term this the "digital bandwidth".

Even limiting the discussion to digital bandwidth only, there are still several distinct concepts to be differentiated:

- **Link Capacity.** The physical speed of a network interface.
- Available Bandwidth. When a link carries cross-traffic (data associated with other users) only the surplus is available for new connections. For example, if the link capacity is 10Mbit/s and the cross traffic is 2Mbit/s, the available bandwidth is
- **End-to-End Bandwidth.** When several links are connected in tandem, the lowest link bandwidth dictates the bandwidth of the overall path. This is sometimes called the "bottleneck" of the connection. The link with the lowest link capacity is called the "narrow link" and the link with the lowest available bandwidth is called the "tight link" [Strauss, J, et al. (2003)].
- **Throughput and Goodput.** The rate at which useful information can be sent. The two terms are often used interchangeably, though "goodput" sometimes indicates

application-layer throughput (the actual throughput minus the segment, datagram and frame headers, and any retransmissions).

- **Effective Bandwidth**. This is the minimum bandwidth that a time-sensitive application needs in order to achieve a required quality of service. (Elwalid and Mitra 1993)

This thesis is mostly concerned with the first three. With these known, throughput/goodput can be calculated without much difficulty. Also the effective bandwidth is more a property of applications than with network behaviour or components, and will also not be addressed.

1.3. Why Measure Bandwidth?

1.3.1. Network Aware Applications

Knowledge of available bandwidth is essential for network-aware applications which adapt their behaviour to the current network status, e.g. controlling outgoing traffic and the sharing of resources to ensure a minimum Quality of Service (QoS). In the presence of a dynamic network load, with the absence of any network-layer reservation mechanism, not only is the accuracy of measurement important but also the speed and frequency of the sampling.

The application actively monitors the performance of the network and adjusts its behaviour accordingly without relying on the underlying network for the Quality of service. Typically it compresses the volume of data in case of high utilisation, reducing the object's size and thus the capacity needed to carry it. If the bandwidth is high, the application will send the data uncompressed and still guarantee delivery within the time limit allocation.

Network-aware applications need to go through three phases to deliver the user-requested media adaptively:

- Monitoring the network connectivity to estimate the available bandwidth in a suitably short time period (ideally seconds rather than minutes).
- Calculating the data that the network can handle before the delivery deadline expires and compressing the fine to fit that constraint.

- Transmitting the compressed file to the user

To sample the bandwidth, the media server sends sample packets to each active node, estimating the available bandwidth by averaging the bandwidth. Unfortunately the estimate is not very accurate since only the long-term historical information is available for the estimation of bandwidth in the immediate future. Also sending continuous probe data in the network can interfere with the other users' traffic. One solution is "on the fly" bandwidth measurement: when a multimedia object is requested, a fraction of the available is spent estimating the current available bandwidth, before adapting and transmitting the actual data. Bandwidth information is therefore "up to date" and there is no unnecessarily probing when no transmission is required. Because time is expended on bandwidth estimation, it presents an overhead which can significantly reduce the time portion for actual data delivery. A major challenge is therefore balancing accuracy, which requires a longer bandwidth testing time, with efficiency which requires a greater transmission time.

1.3.2. Transport Protocols

Transport protocols optimise the rate of transmission to the capacity of the receiver and the intervening network. TCP traditionally uses a sliding window approach: the "window" (i.e. the number of bytes allowed in the network without acknowledgement) increases slowly until losses detected. Then the window size is reduced rapidly and increased slowly again. This is essentially a primitive form of bandwidth probing, in which the bandwidth limit is detected by actual losses, and inevitably involves data loss. It assumes (wrongly in the case of wireless networks) that the only cause of data loss is congestion. Wireless networks also lose significant numbers of packets from bit errors for which TCP will unnecessarily reduce the window size.

An alternative protocol WTCP (Sinha, Venkitaraman et al. 2002) uses a heuristic based upon the average inter-packet separation to detect congestion independently of packet loss, in an equivalent manner to the Packet Pair algorithms described in Chapter 3: if the average packet separation at the receiver is greater than the separation at the sender, this indicates that the available bandwidth of the channel has been exhausted and congestion is occurring. One of the earliest studies was by Mascole et al. (Mascolo, Casetti et al. 2000) who proposed and investigated "TCP Westwood", a TCP extension for wireless

based on available bandwidth measurement by monitoring ACK packets at the sender. Another study led to the proposal of VTP (Video Transport Protocol) which uses a receiver-end bandwidth estimation as part of the flow-control mechanism; when low available bandwidth is detected, the compression of the video is increased so as to lower the bit rate whilst maintaining a consistent frame rate for viewers (Balk, Maggiorini et al. 2003)].

1.4. Chapter Summaries

The remainder of the thesis is organized as follows:

Chapter 2 summarises the wired and wireless Ethernet protocols and compares them with other rival technologies.

Chapter 3 examines and compares the different bandwidth measurement techniques.

Chapter 4 describes the development of an Opnet simulation testbed of packet pair/stream probing and presents baseline experiments involving single wired and wireless links against which more complex scenarios may be compared.

Chapter 5 describes packet-pair probing experiments performed with three network topologies using the Opnet simulation model of Chapter 4. These are a pure wired network, "first mile" wired-cum-wireless, and "last mile" wired-cum-wireless. The results are presented in "raw" form, and as histograms from which dispersion signatures were identified and interpreted.

Chapter 6 explores alternatives to the histogram method for constructing dispersion probability distributions used in Chapter 5, with a view to finding an algorithm to pick out distribution modes (data clusters) automatically, which relate to the particular dispersion signatures.

Chapter 7 utilizes the findings of the earlier chapters in the development of a closed-loop monitoring system for determining the link parameters and traffic load on the network.

Chapter 8 presents an overall summery of the thesis, together with the conclusions and recommendations of the work.

2. WIRED AND WIRELESS ETHERNET

2.1. Introduction

Ethernet (IEEE 802.3) is the most widely used wired technology for connecting computers. Hundreds of thousands of computers have been connected using Ethernet in offices, colleges and other institutions. The cables and hardware required to connect computers are inexpensive, which makes this technology affordable. Wireless versions of Ethernet (IEEE 802.11 Wi-Fi, etc.) are becoming increasingly common in homes and offices. Here computers communicate using microwaves in the 2.4 GHz ISM (Industrial, Scientific and Medical) spectrum which has a bandwidth of about 83 MHz. This chapter outlines the development of both the wired and wireless Ethernet protocols and describes their operating characteristics

2.2. Network Protocols

It is very important to have a set of rules to facilitate the transmission of data, since without rules it would be very difficult for data to travel from one computer to another. The IEEE 802 standards are one set of rules developed and maintained by the IEEE (Institute of Electrical and Electronics Engineers) which helps industry provide advantages such as, interoperability, low product cost, and easy to manage standards. However these standards deal only with the data link layer of the OSI model, which concerns such things as Local Area Network (LAN) and Metropolitan Area Network (MAN) protocols.

2.3. Wired Ethernet

The historical development of Ethernet in its legacy form (CSMA/CD) is outlined in Appendix A, and will not be discussed here. The modern Switched Ethernet moves away from the CSMA/SD mechanism by employing full-duplex links, each of which forms its own separate collision domain. Packets are forwarded by switches which employ a filtering mechanism whereby frames are only forwarded to a port which provides connectivity to the destination host. Thus the other switches and switch ports are left free to perform simultaneous transmission. The result is that each exchange can be carried out at the full interface speed, without collisions and with a noticeable increase in the network bandwidth. This is nowadays the most common form of Ethernet, since a large geographical network can only be built with a hierarchical arrangement of switches.

Network hierarchies were developed to combat problems encountered when networks grow in an unstructured manner without any plan in place. The latter are sometimes referred to as "flat" networks. A hierarchical network typically consists of three layers: a core layer, a distribution layer and an access layer, devices on each layer communicating only with devices on the layer above and the layer below. The IEEE 802.3D Spanning Tree Protocol (STP) enforces a "tree" topology, so redundant links will be closed down and only reactivated in event of failure. Careful design methodology enables one to design a modular topology which is cost effective and efficient and also easy to duplicate as the network grows.

Figure 2.1 UTP Ethernet cable from (Axis-Communications 2010)

Wired Ethernet typically uses a twisted copper wire pairs as a physical transmission medium. An Unshielded Twisted Pair (UTP) cable (Figure2.1) contains a total of eight wires forming four such pairs, terminated by RJ-45 plugs and sockets. The maximum length of a twisted pair cable is 100 m (328 ft.). However, more advanced forms of Ethernet use optical fibre, whose operational length could be anywhere between 10 km to 70 km, though this depends on the type of fibre (single or multi-mode) and light source (laser or LED).

2.3.1. Copper Ethernet

Fast Ethernet (100BaseTX) offers a speed increase to ten times that of the 10BaseT specification while preserving the basic frame format, MAC mechanisms and MTU. Such similarities allow existing 10BaseT applications and network management tools to use Fast Ethernet networks, which slow down to accommodate the slower data rate.

2.3.2. Fibre Ethernet

Data rates of 1,000Mbit/s (1Gbit/s) and over are supported by single and multi-mode optical fibre. 10 Gigabit Ethernet is the latest generation and delivers a data rate of 10 Gbit/s. 10GBaseLX4, 10GBaseER and 10GBaseSR are all based on an optical fibre cable, and can be used to bridge distances of up to 10,000 m (6.2 miles). This kind of Ethernet is mainly used for backbones in high-end applications that require high data rates.

2.4. From Wired Ethernet to Wi-Fi

The TCP/IP layers (Application, Transport, Network, Data Link and Physical) are gradually being adapted for use in wireless environments. However there are some well-known fundamental differences between the wired and wireless paradigms:

Bit Error Rate: This is the probability that a given bit will become corrupted by noise during transmission. It is typically very low in wired media; approximately 10^{-9} in fibre-optics and 10^{-6} in UTP. In a wireless link it is much higher, typically 10^{-3} or 1 error every 1000 bits (Gupta & Kumar 2011).

Bandwidth: In fibre optics this is typically more than 10Gbps, and in UTP it is up to 1Gbps. In a wireless link however, the maximum is 11Mbps and may vary considerably due to channel fading and noise conditions (Tse & Viswanath 2005).

Mobility: The physical movement of end-hosts between regions covered by different networks and access-points creates problems not experienced in wired technology. This can include frequent changes in IP addresses and other problems such as brief loss of connectivity (blackout) and break-up in data transmission during handover (Tse & Viswanath 2005).

Over recent years, much research has been focussed on adapting the Internet structure to accommodate these differences.

2.4.1. Reasons for Wireless Ethernet

Since Ethernet was originally based upon a wireless technology (Alohanet), it can be seen to be returning to its "wireless roots". A wireless network offers advantages and disadvantages compared to a wired network. The advantages are that it is neat and clean, with no untidy cables which can lead to greater flexibility around the home or office within a limited range. On the other hand it has a greater potential for radio interference due to weather, other wireless devices, or obstructions like walls. Wireless networks also suffer poorer noise immunity, and poorer efficiency due to the reintroduction of a common collision domain.

On the whole, wireless LANs offer greater productivity, convenience, and cost advantages over wired networks: Firstly there is the matter of mobility: wireless LAN systems can provide LAN users with access to real-time information anywhere in their organization, which supports productivity and service opportunities not possible with wired networks. In addition, many public places have wireless connection, freeing people from having to be at home or at work to access the Internet. Secondly there is faster installation speed and simplicity: installing a wireless LAN eliminates the need to pull cable through walls and ceilings. Finally there is scalability: wireless LANs can be configured in a variety of topologies to meet the specific needs of applications. These are

easily changed and from small peer-to-peer networks to larger infrastructure networks where thousands of users can roam over an extended area (Stalling 2004).

2.4.2. Taxonomy of Wireless Technologies

Many wireless technologies have been created and new variants appear continually. These can be classified into the same four major categories that are used to classify wired networks, namely: Local Area Networks (LAN), Metropolitan Area Networks (MAN), Wide Area Networks (WAN) and Personal Area Networks (PAN).

LAN's, which operate over the scale of single buildings, were the primary area of deployment for wired Ethernet. The 10 and 100BaseT switched Ethernet protocols are both LAN technologies (though the faster gigabit fibre-based Ethernet has also been used for WAN applications(Fujistu 2006)The protocols for wireless LAN's (WiFi) are specified in the IEEE 802.11 family of standards, which will be discussed in greater detail later on.

PAN technology provides communication over a short distance, typically for devices that are owned and operated by a single user. An example would be connecting wireless headsets, mice and keyboards to a PC. PAN is based upon the IEEE 802.15 family of standards, which provides the basis for Bluetooth, Zigbee and similar technologies.

2.4.3. Local Area Networks and Wi-Fi

The IEEE definition of a Local Area Network (LAN) is: "...a data communication system allowing a number of independent devices to communicate directly with each other, within a moderately sized geographical area over a physical communication channel of moderate data rate"(Alliance 2003). LANs are therefore generally small, fast networks, limited to an area such a building, factory or campus. Larger networks are generally called Metropolican Area Networks (MANs) and Wide Area Networks (WANs).

Wireless LAN technology is generally called Wi-Fi, short for "wireless fidelity". Wi-Fi is technically a trademarked brand name for the wireless standard owned by the Wi-Fi Alliance, (much like Bluetooth, which is trademarked by the Bluetooth Special Interest

Group). In its early stages Wi-Fi technology was nearly always used to connect laptop computers to the Internet, but thanks to its flexibility it is now also found in many non-computer devices such as TVs, digital cameras, and GPS devices.

2.4.4. Metropolitan Area Networks and WiMAX

A Metropolitan Area Network (MAN) serves a role similar to an ISP but for corporate users with large LANs. There are three important features which discriminate MANs from LANs and WANs: Firstly the geographical network size is intermediate between that of a LAN and a WAN. A MAN usually covers an area of between 5 and 50 km diameter, typically the area the size of a city, although in some MANs may be as small as a group of buildings or as large as the North of Scotland (Fairhurst 2001). A MAN (like a WAN) is not generally owned by a single organisation: its communication links and equipment are generally owned by either a consortium of users or by a single network provider who sells the service to the users. The level of service provided to each user must therefore be negotiated with the MAN operator and performance guarantees are normally specified. Finally a MAN often acts as a high speed network to allow the sharing of regional resources (similar to a large LAN). It is also frequently used to provide a shared connection to other networks using a link to a WAN. A typical use of MANs to provide shared access to a wide area network, as shown in Figure 2.2.

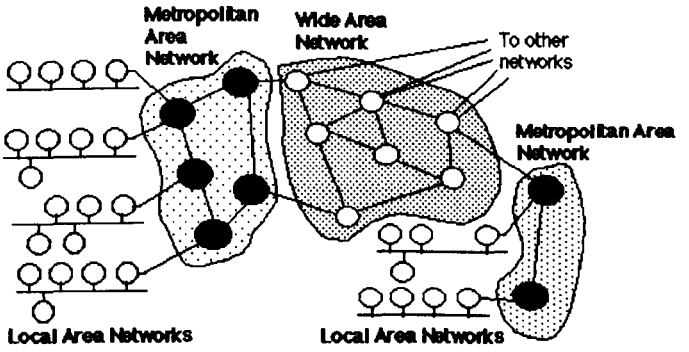


Figure 2.2 Use of MANs to provide regional networks which share the cost of access to a WAN from (Fairhurst 2011)

Wireless MAN functionality is provided by WiMAX, a wireless digital protocol whose standards are specified by the IEEE 802.16 working group. WiMAX provides broadband wireless access (BWA) up to 30 miles (50 km) for fixed stations, and 3 - 10 miles (5 - 15

km) for mobile stations. In contrast, the WiFi/802.11 (wireless LAN) standard is limited in most cases to only 100 - 300 feet (30 - 100m).

With WiMAX, WiFi-like data rates are easily supported, but the issue of interference is lessened. WiMAX operates on both licensed and non-licensed frequencies, providing a regulated environment and viable economic model for wireless carriers. WiMAX can be used for wireless networking in much the same way as WiFi, but also allows more efficient bandwidth useage, interference avoidance and higher data rates over longer distances.

2.4.5. Wide Area Networks (WANs)

The term Wide Area Network (WAN) usually refers to a network which covers a large geographical area using leased communications circuits from telephone companies or other communications carriers. Transmission rates are typically 2, 34, 45, 155 and 625 Mbps, or sometimes considerably more. WAN applications include public packet networks, large corporate and military networks, banking and stock brokerage networks and airline reservation networks. Some WANs span the entire globe, but most do not provide truly global coverage.

Organisations supporting WANs using IP are known as Network Service Providers (NSPs), and form the core of the global Internet. By connecting the NSP WANs together using links at Internet Packet Interchanges (sometimes called "peering points") a global communication infrastructure is formed. However, except for major corporate clients, these NSPs do not generally handle individual customer accounts but instead deal with intermediate organisations who they can charge for high capacity communications. They generally have an agreement to exchange certain volumes of data at a certain "quality of service" with other NSPs. Practically any NSP can reach any other NSP, but may require the use of one or more other NSP networks to reach the required destination. NSPs vary in terms of the transit delay, transmission rate, and connectivity offered. (Figure 2.3.)

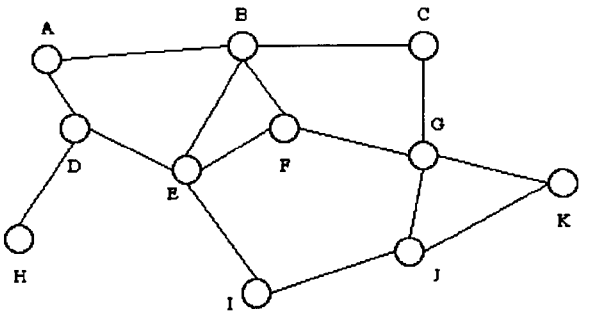


Figure 2.3 Typical "mesh" connectivity of a Wide Area Network from (Fairhurst 2011)

Wireless WAN (WWAN) services are typically delivered to smart phones and other handheld devices sold by cellular service providers, but other mobile devices can also use them; for example, some netbooks have WWAN cards installed. Unlike Wi-Fi cards which can be used with just about any Wi-Fi hotspot, WWAN devices are provisioned to access to specific service providers' network. WWAN technologies include GSM/UMTS, CDMA One/CDMA2000, and are expected to become increasingly available with the rise of 4G technologies.

This section has briefly described the main branches of wireless technology, in order to put Wi-Fi in its technological context. However, from this point onwards the thesis will concentrate exclusively on Wi-Fi as a wireless technology, and its use in conjunction with wired Ethernet in wired-cum-wireless networks.

2.5. WiFi

2.5.1. Infrastructure and Ad Hoc Modes

WiFi can be configured to work in two different modes: an "infrastructure" mode in which networked devices communicate via a common access point (AP) or wireless hotspot, and an "ad-hoc" mode in which they communicate either directly or via intermediate host devices which behave as routers. Thus an ad-hoc network is a local area network (LAN) built spontaneously as devices connect, instead of relying on a base station to coordinate the flow of messages, where the individual network nodes forward packets to and from each other. (In Latin, *ad hoc* literally means "for this," i.e. "for this special purpose" and thus by extension, "improvised" or "impromptu".)

2.5.2. Physical Layer Characteristics

Wireless communication applies across a wide range of network types and sizes, and governmental regulation limits radio frequency (RF) spectrum available for this type of communication. A licence is required to transmit in some part of the spectrum while other parts remain unlicensed. Wireless computer networks commonly use the Industrial, Scientific and Medical (ISM) group of bands, though there are local laws which set frequency allocations which differ from country to country. Maximum allotted transmission power and location (indoor, outdoor) are also determined by the same laws.

The normal range of a wireless radio network depends on the emission power, and is typically 10–100 meters up to 10 km per machine. It also depends on other factors like the data rate, the frequency and the type of antenna used. There are several types of antennas including omnis (omnidirectional), sector antennas (directional), yagis, parabolic dishes, or waveguides (cantennas). Infrared transmission is another very popular medium of wireless transmission, though opaque materials cannot be penetrated and the range is limited to about 10 meters. This is why infrared technology is mostly used for small devices in WPANs (Wireless Personal Area Networks) for connecting PDAs to laptops inside a room.

There are three main differences between wired and wireless channels. Firstly in wireless channels radio waves experience free-space attenuation; signal levels become rapidly weaker as the transmitter recedes since energy is spread over a much larger area. This is contrasted with wired channels in which the signal attenuation (mostly due to electrical losses) is much more gradual. Other wireless propagation effects are also important such as occlusion from objects which the waves cannot penetrate and diffraction around the corners of such objects extending the line-of-sight range.

In addition to this, multiple channels must share a common radio medium, just as legacy Ethernet stations share a common bus. However, due to the aforementioned signal attenuation process collisions are difficult to detect since a transmitting station's own signal usually drowns out the transmissions of other stations. For this reason an alternative to CSMA/CD has been developed, called CSMA/CA (the CA standing for "collision avoidance"). Wireless transmissions are also much more prone to noise than

wired transmissions and for this reason (as well as the collision problem) the mechanisms for error detection, acknowledgement and retransmission are of great importance.

Standard	Data Issued	Available Bandwidth (MHz)	Unlicensed frequency of operation (MHz)	Data rate per channel (Mbps)	compatibility
802.11	1997	83.5	2.4 to 2.4835DSSS,FHSS	1,2	802.11
802.11a	1999	300	5.15 to 5.35 OFDM 5.72 to 5.825 OFDM	6,9,12,18,24,36,48 and 54	Wi-Fi5
802.11b	1999	83.5	2.4 to 2.4835DSSS	1, 2 ,5.5 and 11	Wi-Fi
802.11g	2003	83.5	2.4 to 2.4835DSSS, OFDM	1, 2 ,5.5, 6, 9, 11, 12, 18, 24, 36,48and 54	Wi-Fi at 11 Mbps and below

Table 2.1 IEEE 802.11 physical layer standards.

Table 2.1 shows a list of 802.11 physical layer standards which can be outlined as follows: While the original 802.11 used the ISM (2.4 and 2.5GHz) band, 802.11a took advantage of the newly unlicensed 5MHz U-NII (Unlicensed-National Information Infrastructure) spectrum created by FCC to enable users receive fast wireless internet. This has a much greater bandwidth (300MHz, compared with the 83MHz of ISM), though it was incompatible with existing 2.4GHz equipment. Per-channel data rates of 6, 9, 12, 18, 24, 36, 48 and 54 Mbps are possible in IEEE 802.11a.

IEEE 802.11b is an extension of 802.11 DSS Scheme which provides data rates of 5.5 and 11 Mbps. The same 11MHz bandwidth is required by each channel, and a modulation scheme called "complementary code keying" is employed to achieve a higher data rate in the same spectral bandwidth. Unlike 802.11a it uses the 2.4MHz ISM spectrum.

The most common WiFi standard used today is IEEE 802.11g. This also uses the ISM spectrum, but data rates are extended to between 12 and 54Mbps per channel. IEEE 802.11g is compatible with 802.11b for the simplest of reasons that they both operate in the 2.4GHz range.

2.5.3. Link-Layer Characteristics

As with wired Ethernet, the placement of bits onto the physical medium is regulated by the Media Access Control (MAC) mechanism which together with the LLC layer forms the link-layer protocol. In the case of IEEE 802.11 (WiFi) the MAC usually has two sub layers: the lower layer is the Distributed Coordination Function (DCF) which is responsible for providing access to the physical layer. The upper layer is the Point Coordination Function or PCF (see Figure 2.4).

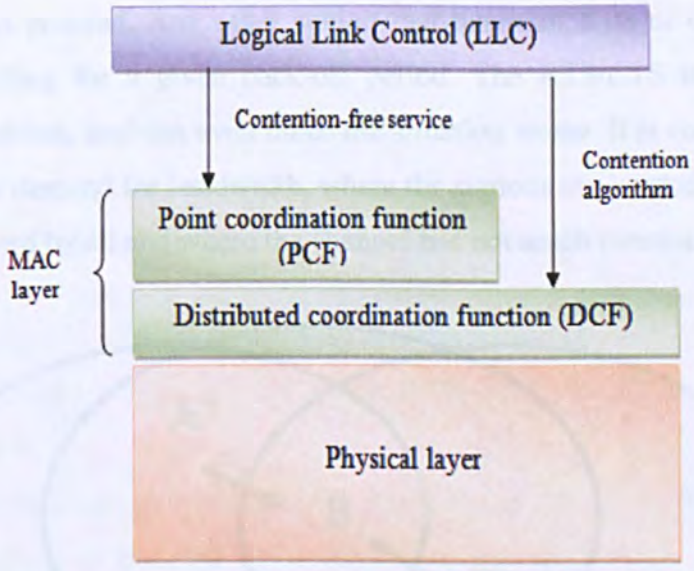


Figure 2.4 IEEE 802.11 link layer architecture.

MAC Functionality is considered to be very reliable and offers several benefits, it is very reliable and considered safe for data transfer. It offers controlled access to the shared wireless medium; it also protects the data that it has delivered. All in all it is a safe and a reliable way of data transfer. The MAC Frame Protocol on the other hand is very different from MAC functionality, it is very noisy and considered unreliable when it comes to data transfer. It has several hidden node problems, all stations are required to participate, and every station must react to every frame it receives. It has several other problems too.

2.5.4. The Hidden Node Problem

This is illustrated in Figure 2.5, where nodes A, B and C communicate in a wireless LAN. Although stations A and C are within range of the access point B, they are outside range of each other. If A wishes to transmit to B while C is transmitting to B, A will believe the medium to be free and will therefore send. Thus the CSMA/CA mechanism fails, and there is a possibility of frames getting corrupted. This problem is addressed by means of the RTS (Request To Send) and CTS (Clear To Send) frames shown in Figure 2.6. A node wishing to send data first sends an RTS packet, and the receiver replies with a CTS packet telling it to proceed. Any other station that hears an RTS or CTS transmission refrains from sending for a given back-off period. The RTS/CTS is not a complete solution to the problem, and can even make the situation worse. It is commonly disabled where there is low demand for bandwidth, where the stations are located in an area where transmission is heard by all and where the channel has not much contention.

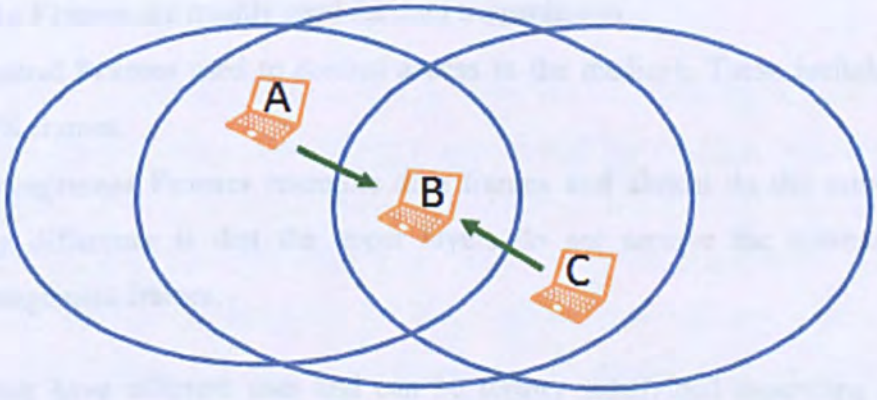


Figure 2.5 Hidden node problem

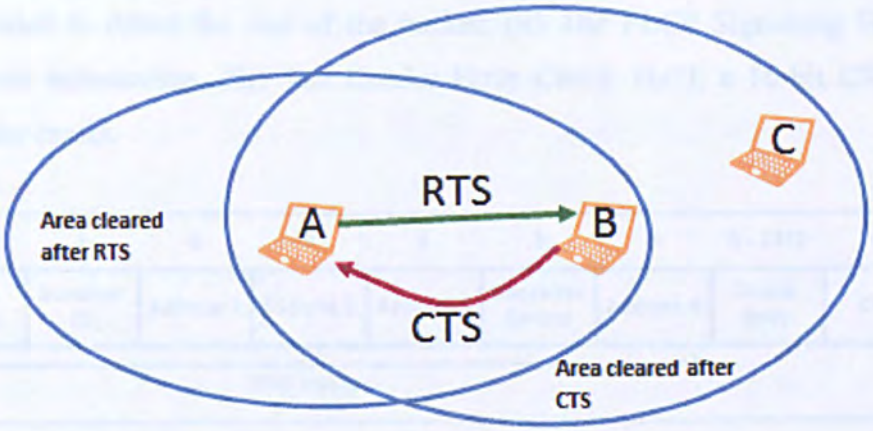


Figure 2.6 RTS and CTS address the hidden Node Problem

2.5.5. Acknowledgements and Re-transmissions

There are three main types of frames:

- **Data Frames** are mainly used for data transmission.
- **Control Frames** used to control access to the medium. These include CTS and ACK frames.
- **Management Frames** resemble data frames and almost do the same job. The only difference is that the upper layers do not receive the information from management frames.

These frames have different uses and can be further subdivided depending upon their specific functions. The 802.11 frame header consists of the following components: The Preamble, the PLCP (Physical Layer Conversion Protocol), the MAC data and the CRC.

The preamble depends on the specific physical layer protocol, and it includes the following components: (i) SYNCH is an alternating sequence of zeros and ones, used to synchronise the receiver hardware to the transmitter clock signal. (ii) SFD (start of frame delimiter) consists of 16-bit binary pattern 0000 1100 1011 1101 which indicates the end of the preamble and the beginning of the header proper.

The PLCP Header contains the following logical information which the physical layer uses: (i) The PLCP_PDU Length Word, specifying the number of bytes in the packet.

This is needed to detect the end of the packet. (ii) The PLCP Signalling Field which specifies rate information. (iii) The Header Error Check Field, a 16-bit CRC field to detect header errors.

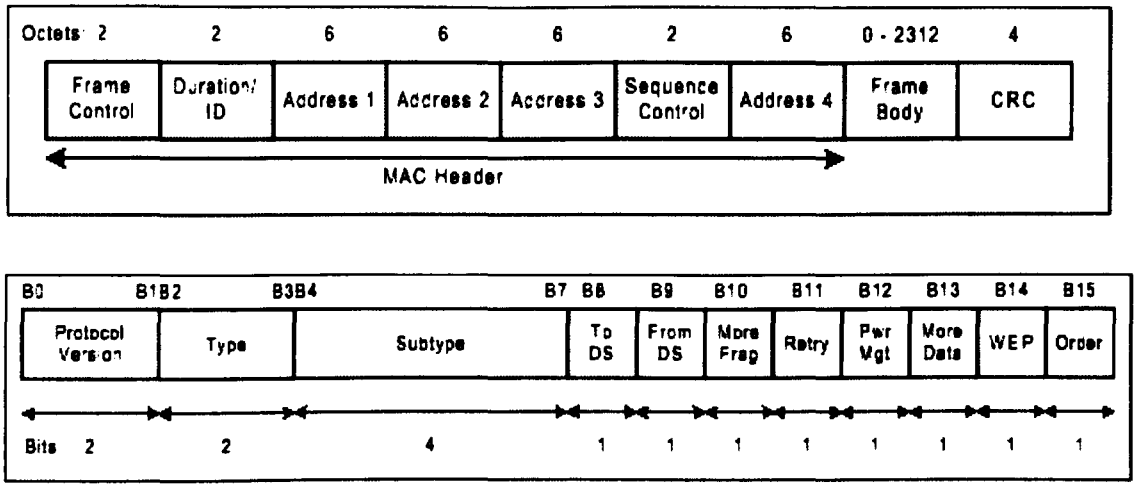


Figure 2.7 Mac frame format from (Brenner 1997). The lower figure expands the 2 octets of the frame control field.

The MAC frame format is shown in Figure 2.7. The first two octets form the Frame Control Field, which is expanded in the lower part of the diagram. The first two bits identify the type of 802.11 version used, and the Type and Subtype identify the frame type. (Data, ACK etc.) The ToDS flag is set to 1 if the frame is addressed to the AP for distribution to the distributed system. Similarly the FromDS flag is set when the frame comes from the distributed system. More Fragments is set when there are more fragments of the same frame other than the current fragment, and Retry (as the name suggests) indicates when the process of fragmentation is being retried. Power Management indicates the power management mode the station will revert to after the frame is done with transmission, and More Data means there are more frames that are buffered to the station. WEP indicates that the data is encrypted using the WEP algorithm, and Order that the frame is being sent by the strictly-ordered service class. Finally the Duration/ID flag can have several different meanings, depending on the frame type.

Unlike the 802.3 Ethnernet, 802.11 frames have four address fields. Address 1 is always the destination address and Address 2 the transmitter address, while Addresses 3 and 4 have more specialised meanings which depend on the specific values of the ToDS and

FromDS flags. The sixteen Sequence Control bits specify the required order for different fragments which belong to the very same frame, and is responsible for identifying packet duplication. Finally CRC contains a 32-bit error-check field based on the Cyclic Redundancy Check algorithm.

The RTS and ACK frame formats are similar but much simpler, and are shown in Figure 2.8. The RA and TA fields specify the transmitter and receiver addresses, and the other fields have the same meanings as those in the data frame described above.

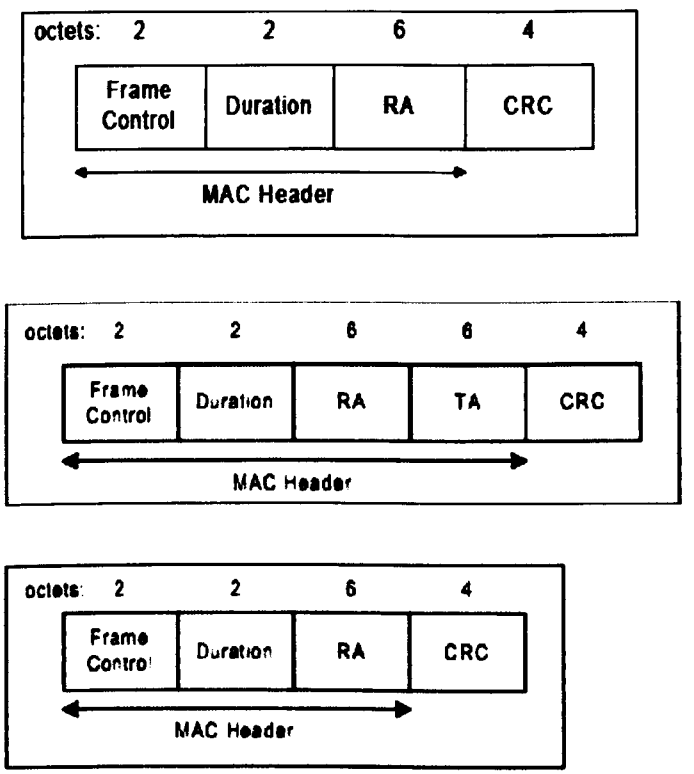


Figure 2.8 RTS, CTS and ACK frame formats from (Brenner 1997)

The IEEE 802.11 does three main tasks properly and they are as follows:

- Authentication
- Key Management
- Data transfer privacy

These are some security issues which are well handled by IEEE 802.11, the WPA and the WEP key options provide complete security to the users. Leaving a wireless network unprotected is never advisable, anyone can access an open network and retrieve critical information like credit card details, banking passwords and other private information (Bing 2008).

2.6. Wireless Challenges

Over recent years, much research has been focussed on adapting the wired Internet structure to accommodate the specific issues of wireless. The following issues have been identified:

Inappropriate TCP Response. Many of the phenomena experienced in wireless networks (frequent bit errors, variable latency, temporary blackout) may be misinterpreted as arising from congestion – the only major source of loss in a wired network. Traditional TCP responds by reducing its congestion window (i.e. the maximum allowed number of unacknowledged packets), causing a further unnecessary disruption of service (Pentikousis 2000) **Handovers.** The handover process may cause data losses and a drastic change in the quality of service provided, when the user moves from an idle to a busy service area (Dubois, Sorensen-Oumer et al. 2005)

Bandwidth Management Issues. Traditional techniques for the monitoring and allocation of available bandwidth assume a simplistic point-to-point link model which may break down under the conditions experienced in wireless networks. These may include non-FIFO scheduling, imprecisely-defined link rates and a long variable latency, independent of packet size(Johnsson, Melander et al. 2006)

Resource Allocation and Scheduling. The demand for multimedia data-streaming applications for mobile users requires per-user rates of 100kbit/s and above. The scheduling of packets in the wireless channel must give priority to users capable of supporting these high rates, and to the relative importance of individual packets' contents (Pahalawatta, Berry et al. 2007)

It is easy to see that there is a close interrelationship between these four areas: The development of new transport-layer algorithms requires a reliable model of the wireless

channel behaviour, as do techniques for the estimation of bandwidth. Resource allocation also depends on the ability to estimate bandwidth, particularly under the fast-changing situations imposed by handovers and blackouts.

It was therefore decided to concentrate on the measurement and management of bandwidth and investigate how this impacts upon the other areas. Available bandwidth in wireless environments is still poorly understood, and considerable research is needed in this areas.

In one of the earliest studies on the subject, Pentikousis (Pentikousis 2000) proved that traditional TCP, though effective in the wired environment, has a very poor performance on wireless networks. Moreover, solutions devised for wireless LANs do not perform well in wireless WANs, and vice versa. One of the major dilemmas is the control of the congestion window, which traditionally narrows as packet-losses are detected. Though this eases congestion, it is not beneficial when losses are caused by bit-error corruption. Thus efficient flow-control requires corruption losses to be distinguished from losses caused by buffer-overflow.

Study by Biaz and Vaidya (Biaz and Vaidya 1999) proposed a simple scheme which enables a TCP receiver to distinguish congestion losses from corruption losses, by monitoring the inter-arrival gap. The scheme works in the case where the last hop to the receiver is a wireless link, and also the bottleneck. This mechanism was added to TCP-Reno in order to compare its performance against traditional TCP-Reno and a hypothetical "Ideal-Reno" with a perfect knowledge of the causes of loss. The proposed scheme performed similar to ideal scheme, but only when the congestion and error-rate were low.

Fu et al. (Fu, Greenstein et al. 2002) designed and implement a TCP-compatible transport protocol for ad hoc networks based on end-to-end measurements. This technique reduced problem of false detection through the use of multiple heuristics, and thus improving the transportation performance in TCP-friendly way.

Leung et al. (Leung, Klein et al. 2004) proposed and studied two effective ways to improve TCP throughput in wireless networks:

- Select a retransmission timeout (RTO) threshold higher than the de facto standard. Simulation proves a significant reduction in timeout, and provides a relative throughput gain up to 13.7% (based on RTT measurements in a commercial 3G network and in a simulated network environment).
- Appropriate use of selective repeat (SR) and go-back-N (GBN) as retransmission policies upon packet timeout. Significant performance gains have been shown for both methods, based on simulated and measured RTT traces in commercial 3G networks.

Klein et al. (Leung, Klein et al. 2004) have detected the presence of delay spikes in wireless networks, and described their negative impact on TCP performance by causing false timeouts. Rather than modifying the TCP protocol, this paper investigated an new methodology that reduces the probability of TCP timeouts and increases the TCP throughput performance. This methodology is to inject false delay in the round-trip path in order to increase the variation of the round trip estimate, and thus increase the timeout threshold calculation.

Sinha et al. (Sinha, Venkitaraman et al. 2002) presented a new protocol WTCP (Wireless-TCP), a reliable transport protocol that addresses rate control and reliability over commercial WWAN networks such as CDPD. WTCP is rate (rather than sliding-window) based, uses only end-to-end mechanisms, performs rate control at the receiver rather than the transmitter, and uses inter-packet delays as the primary rate control metric. WTCP has been implemented and evaluated over the CDPD network, and also simulated using the *ns-2* software. Results indicate that WTCP can outperform comparable algorithms such as TCP-NewReno, TCP-Vegas, and Snoop-TCP by between 20% to 200% under typical operating conditions.

Further work was performed by Cheng et al. (Cheng, Cheng et al. 2000) , who expanded the features of WTCP to include fast acknowledgement, on-demand retransmission, a K-success status report and a time-out mechanism at the receiver module. At the sender side, this WTCP is able to tackle problems caused by the vulnerable wireless environment and TCP flow control mechanism.

A large number of applications make use of multimedia streaming, with data rates in access of 100kbit/s per user. This is real-time data, which is highly susceptible to latency

and/or jitter and therefore needs a higher Quality-of-Service (QoS) than time-insensitive data (such as HTTP). Although QoS mechanisms such as *Diffserv* and *Intserv* exist for this purpose, they have proved ineffective in the erratic world of wireless communications (Joanna 2007).

The general aim is to maximise throughput while maintaining fairness between users. However, there are two complications to this: Firstly, not all of the packets in a video sequence are of equal importance, and an increase in throughput does not necessarily bring an increase in quality. (This has led to the development of cross-layer algorithms for scheduling packets according to their relative importance (Pahalawatta, Berry et al. 2007) Additionally, wireless networks suffer from the particular problem of channel fading: Users' data-rates depend on their respective fade-status, and those experiencing heavy fading cannot support the data rates required for streaming. Resources must therefore be de-allocated from such users, and given to users who can support the required rate (Pahalawatta, Berry et al. 2007).

2.7. Comparisons with Other Technologies

There are other wired and wireless technologies which are used as alternatives to the Ethernet-based technologies described above. Although they are not addressed in this project, for the sake of completeness some of them are discussed briefly below.

2.7.1. Wired Technologies

Frame relay, like Ethernet relays data units called frames using permanent virtual circuits (PVCs) which are provided by the network. Like Ethernet (and unlike schemes such as Time Division Multiplexing) the customers need not pay for full time leased lines. Each frame has its intended destination to which it travels and the charges are levied purely depending upon usage. Another feature lets users assign higher priority to some frames than to others. Frame relay technology is inspired by the older X.25 technology which employed packet-switching and was specifically designed in order to transfer analogue data like voice conversations. Frame relay is however much faster and cost-effective. It favours speed over reliability, and makes no attempt to correct errors. The end points are responsible for retransmitting dropped frames.

In the 1990s the Asynchronous Transfer Mode (ATM) was considered the "next big thing" which would ultimately replace the Internet for end-to-end communications. Many people believed it would solve all the major bandwidth problems, though others were doubtful. ATM is a "slotted time" protocol which uses 53 byte "cells", typically transmitted at 155Mbits/s. The four transmission options for ATM are (in order of decreasing priority) CBR, VBR, ABR and UBR. CBR stands for constant bit rate, VBR stands for variable bit rate, ABR stands for available bit rate and lastly UBR stands for unspecified bit rate.

2.7.2. Wireless Technologies

Other forms of wireless have already been discussed, but are revisited here for completeness. Bluetooth and Infrared technologies are used for short distances, though the latter has a plethora of drawbacks. It is incompatible with many devices and it is comparatively slow and time consuming. Bluetooth on the other hand still works if objects are placed between the communicating devices. Also, Bluetooth uses a standard 2.4GHz frequency so that all Bluetooth-enabled devices will be compatible with each other. The only drawback of Bluetooth is that because of its high frequency its range is limited to 30 feet, and connectivity is usually limited to a single room. However, the short range can be seen as a positive aspect since it adds to security.(TechTerms 2012).

Zigbee is a specification for wireless personal area networks (WPANs) operating at 868 MHz, 902-928 MHz, and 2.4 GHz. (A WPAN is a personal area network, a network for interconnecting an individual's devices, in which connections are wireless.) Using ZigBee, devices in a WPAN can communicate at speeds of up to 250kbits/s, while physically separated by distances of up to 50 meters. ZigBee is based on the IEEE 802.15 standard. ZigBee provides for high data throughput in applications where the duty cycle is low, amking it ideal for home, business and industrial automation where control devices and sensors are commonly used. Such devices operate at low power levels which, in conjunction with their low duty cycle (typically 0.1 percent or less) leads to long battery life. Applications well suited to ZigBee include heating, ventilation, and air conditioning (HVAC), lighting systems, intrusion detection and fire sensing. ZigBee is compatible with most topologies including peer-to-peer, star network, and mesh networks and can handle up to 255 devices in a single WPAN (TechTarget 2005).

Finally Wi-Max was developed by the WiMAX Forum as one of the technologies for 4G networks. It can be used in both point-to-point and typical WAN type configurations that are also used by 2G and 3G. Its formal name is IEEE 802.16 (Tapia 2009).

2.8. Summary

This chapter presented a brief summary of wired and wireless Ethernet and some of the major technological issues associated with them. The following topics were addressed:

- The historical background of Ethernet from its origin as Alohanet.
- The Legacy Ethernet protocol with the CSMA/CD mechanism.
- The Switched Ethernet protocols, both wired and fibre-based.
- Wireless Ethernet, including WiFi and WiMAX, their operation and security features.
- Comparison with other technologies.

3. BANDWIDTH MEASUREMENT TECHNIQUES

3.1. Introduction

Bandwidth measurement is an important process at the transmitter end of a path because it allows the user to avoid congestion by selecting a suitable rate of transmission for acceptable QoS and to adapt file compression ratios to match the available bandwidth.

The bandwidth of a path through the network is usually determined by its tightest bottleneck, i.e. the transmission rate of the slowest forwarding element between the sender and receiver. This could refer to the lowest forwarding capacity in the path (the "narrow link") but a connection is seldom alone on a path. It shares hops with cross traffic from other sources. Thus the "available bandwidth", the smallest *surplus* bandwidth on the path (the "tight link") which the application can access when sharing hops with cross traffic, is also of major interest. Bottleneck and available bandwidth are both extremely important, as each is responsible for capturing relevant properties of a network. For short time-scale processes (e.g. compression of objects to optimal size for transmission) the available bandwidth is of greater importance, while longer time-scale processes (such as admission control and server selection) may be helped by both measures. Other network applications (such as capacity provisioning) are concerned mostly with bottleneck measurement.

In practice the available bandwidth depends on many factors including the application itself, the network protocols, the characteristics of the cross traffic and the routers. Cross traffic varies in time, making the measurement of available bandwidth an even more elusive goal. However, three metrics have been identified to characterize the available bandwidth of a path:

- **The Proportional Share.** The queuing policy varies between networked components, but in a simple first come first serve (FCFS) system, each component

gets a share of the bandwidth proportional to its rate. Most probing theories are based on this idea.

- **The Surplus.** This is loosely speaking the amount of unused bandwidth, but the exact definition is problematic. In an early paper Melander et al. defined it as "...the highest possible sending rate of a new sender without affecting cross traffic on the path"(Melander, Bjorkman et al. 2000). However, packets in different streams may interact (queue behind each other) even before surplus is exhausted, and thus affect each other's delay performance. However, attempts to exceed surplus will lead to sustained queue growth (causing greatly increased delays) until packets are dropped.
- **The Protocol Dependent Available Bandwidth.** If cross traffic is TCP-controlled, then attempts to exceed surplus will cause congestion window sizes to be cut, and bandwidth freed. When this happens the surplus will increase, as a result of the new traffic(Melander, Bjorkman et al. 2000).

Many specific tools are available such as Pathload (Suthaharan and Kumar 2008) and Pathchirp (Ribeiro, Riedi et al. 2003). However the fundamental approaches adopted by these tools are more limited in number. While some tools rely on the cooperation of network devices, others are truly end-to-end and rely only on functionality implemented in the receiver and the sender. The former are clearly incapable of detecting certain types of devices, e.g. tools relying on IP-layer functionality will not usually recognise Ethernet switches. Only if the network is properly functioning and all objects are able to show themselves, can the complete topology of the path be determined. With the expanding size and speed of internet these conditions are now rarely met. Furthermore some network administers may block access to components' diagnostic tools because of the fear of malicious attack.(Castro, Coates et al. 2004) End-to-end probing on the other hand has all its functionality at the end-stations. Several fundamental approaches have been developed, which are summarised in Table 3.1. These will be discussed in the remainder of this chapter.

	Link Capacity	Available Bandwidth
Per Link	Iterative TOPP/VPS	Iterative TOPP/Idle Rate
End-To-End	PPTD	SloPS/TOPP

Table 3.1 Classification of Bandwidth Measurement Tools.

3.2. Intermediate Device-Dependent Methods

Several approaches to network monitoring make use of functionality within the intermediate devices between the endpoints. This functionality could belong to the IP layer of a router or layer-3 switch, or alternatively be application layer protocols such as SNMP. Some examples are given below.

3.2.1. Variable Packet Size (VPS) probing

VPS probing measures the RTT from the source to each hop of the path as a function of packet size, forcing packets to expire at a particular hop using the Time-To-Live (TTL) field of the IP header (Prasad, Dovrolis et al. 2003). Routers thus discard probing packets and return ICMP "time-exceeded" error messages to the source, which is used to measure the RTT of that hop. There are five delay components in the RTT:

- **Serialization Delay** of packet size L (bits) at a link of transmission rate C (bits/s) is equal to L/C, and is thus proportional to packet size.
- **Propagation Delay** is the time for each bit of the packet to traverse the link, and is independent of the packet size.
- **Inter-frame Spacing** is the time delay enforced by each router between successive frame transmissions, and is independent of packet size.
- **Queuing Delays** occurs in router and switch buffers when there is contention at the input or output ports of these devices.
- **Processing Delay** for the router to delete the time expired packets and produce

the IMPC message.

The sending host transmits probing packets of a given size to each layer-3 device along the path. Ideally at least one of these packets (together with the ICMP reply that it generates) encounters no queuing delay, so the minimum RTT for each packet size has two components: a delay that is independent of probe packet size (i.e. the combined inter-frame and propagation delays for the probe and ICMP packets and the serialization delay for the constant-sized ICMP packets) and a serialisation delay proportional to probe packet size. Since the probe packet size is known, the latter allows the transmission rate C to be computed. One drawback of VPS probing is that store-and-forward switches also introduce serialization delays, but they do not generate ICMP TTL-expired replies because they are not visible at the IP layer.

3.2.2. SNMP-Based Measurements

Of the five Network Management functions specified by ISO, we are (for the purposes of bandwidth measurement) mostly concerned with Performance Management. Simple Network Management Protocol (SNMP) can obtain a great deal of performance information from agents on managed devices, mainly to facilitate the work of IT administrators who must act quickly to maintain consistent quality of service in the event of failure. This information includes such things as interface speed and the numbers of packets and bits sent and received, from which the link utilisation (and hence the available bandwidth) may be computed. Unfortunately this requires administrator access to all SNMP agents in a monitored path, and not all devices have SNMP agents implemented on them.

3.3. Methods Relying on One-Way Latencies

Most end-to-end techniques (e.g. SLoPS, TOPP and Spruce) rely upon the fact that temporary congestion can be observed when a short-lived stream of probe packets is sent. Congestion is usually accompanied by the increasing end-to-end latency of successive packets. If these latencies can be measured, delay-trends (increasing, remaining constant or decreasing) can be reported by the receiver.

A typical example of this is the Self-Loading Periodic Stream (SLoPS) technique (Prasad, Dovrolis et al. 2003). The source transmits a stream of equal-sized packets at a certain rate R bits/s and monitors variations in their one-way delay. If R is greater than the available bandwidth A, the stream causes a short term overload and the one-way delays increases. On the other hand, if the stream rate $R<A$ the packets create no backlog and the average one-way delays remains constant. The sender uses an iterative binary search algorithm to identify the approximate value of A, probing the path at different rates while the receiver notifies the sender of the resulting delay-trends. The sender also ensures that the network carries no more than one stream at any time, that the average probing rate is less than 10% of the available bandwidth, and that a silent period exists between successive streams. The available bandwidth estimate A may vary during the measurements, but SLoPS can detect this when the one-way delays of a stream fail to show a clear increasing or non-increasing trend. In such cases the methodology reports a "grey" region.

A major drawback of one-way latency measurement is that it requires the probe-packet's timestamp, derived from the transmitter's clock, to be compared with the local clock at the receiver. If the two clocks are significantly out of synch, then errors of delay measurement may result. For this (and other) reasons, many workers have abandoned the measurement of absolute latency, and have concentrated instead upon the dispersion (or time difference) between successive probe packets.

3.4. End-to-End Dispersion-Based Methods

Packet dispersion techniques make use of the "bottleneck spacing effect" of FIFO (first-in-first-out) queuing networks, which governs the difference in arrival times of two packets of the same size travelling from the same source to the same destination. This can potentially be used to determine the "narrow link" (the smallest link capacity on the path) and the "tight link" (the smallest surplus or available bandwidth). The bottleneck spacing effect as described below.

3.4.1. Bottleneck Spacing Effect

Here we consider the network in the absence of cross-traffic, under which condition the narrow link and the tight link refer to the same "bottleneck". Figure 3.1 shows the basic phenomenon: Closely spaced packets may be "squashed together" when passing through a tight bottleneck, causing their separation to be greater after the constriction than before. To express this mathematically, then basic equation governing dispersion is

$$\Delta_{out} = \max(L/C, \Delta_{in}) \tag{3.1}$$

where Δ_{in} and Δ_{out} are the dispersions before and after the bottleneck. This is illustrated in Figure 3.2, where the vertical axis represents queued bits and the horizontal axis time. (Note that Eqn. (3.1) is a simplified version which ignores the inter-frame spacing.)

$$C = L/\Delta_{out} \tag{3.2}$$

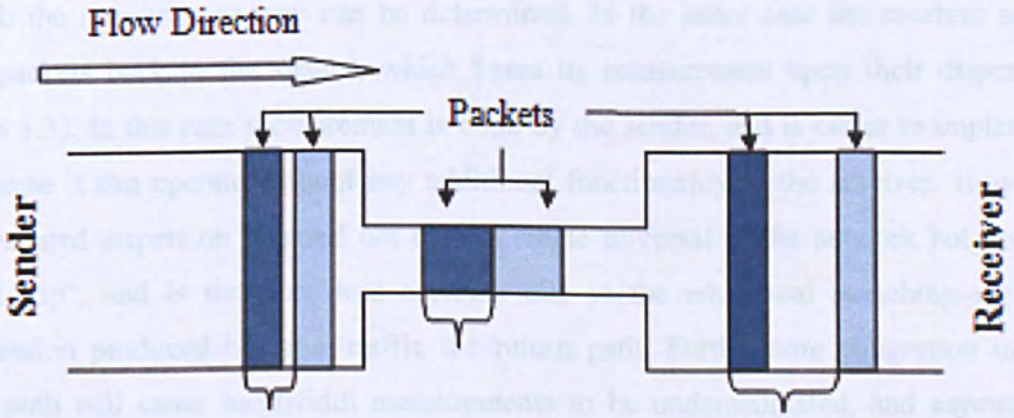


Figure 3.1 Illustration of bottleneck spacing effect (Balakrishnan, Padmanabhan et al. 1997)

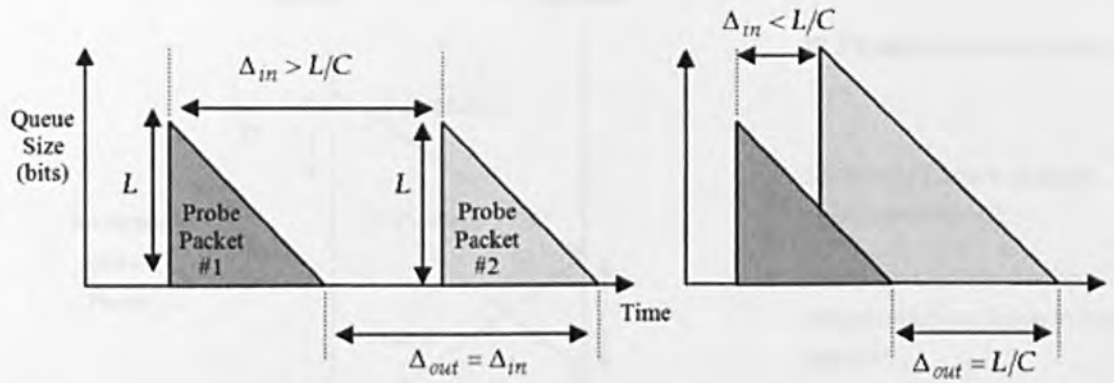


Figure 3.2 Illustration of bottleneck spacing effect from (Tunnicliffe 2010).

3.4.2. Sender and Receiver-Based Methods

Whether a technique is "sender based" or "receiver based" depends upon the functionality devolved to the sender by the sender who is measuring the bandwidth. In the former case, the dispersion between packets is measured at the receiver, where the effect of passing through the network one-way can be determined. In the latter case the receiver sends ACK packets back to the sender, which bases its measurement upon their dispersion (Figure 3.3). In this case measurement is done by the sender, and is easier to implement as because it can operate without any additional functionality in the receiver. However the measured dispersion is based not upon a single traversal of the network but upon a "round trip", and is therefore less accurate due to the additional stretching-out and compression produced by cross-traffic the return path. Furthermore congestion in the return path will cause bandwidth measurements to be underestimated, and asymmetry between the sending and receiving bandwidths will complicate the picture further.

We therefore assume that the receiver collects the timing information from probe packets, and is therefore a receiver-based packet pair (RBPP) system (Figure 3.4). Since it knows what the separation was at the transmitter, the receiver is aware when the packet pair dispersion is not stretched out by network and is able to reject such pairs from the bottleneck calculation.

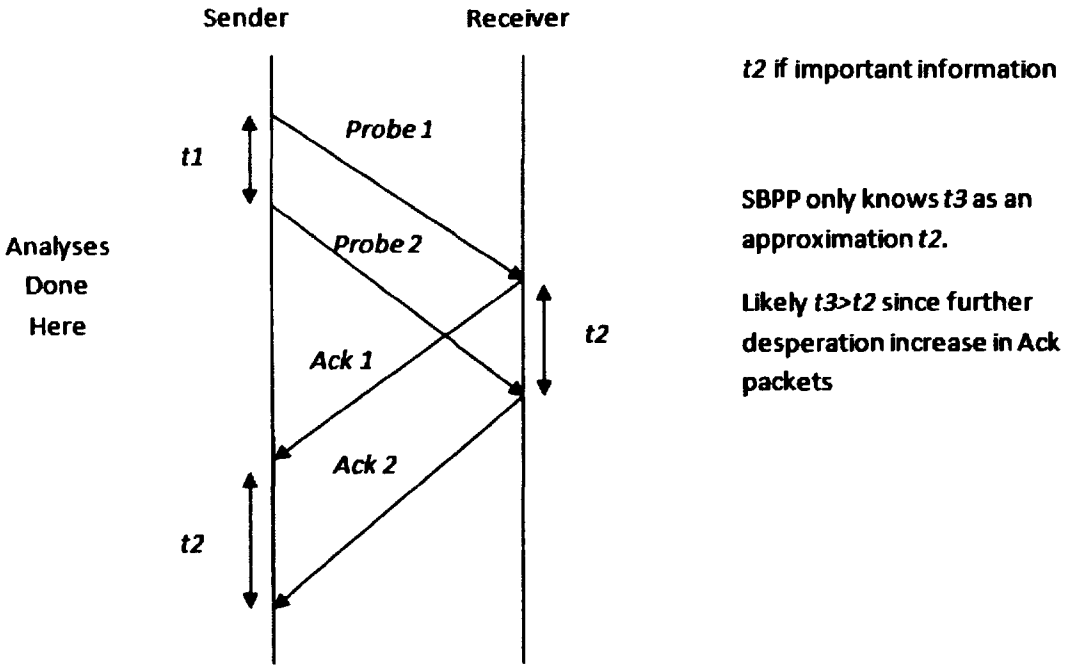


Figure 3.3 Sender-based packet pair (SBPP) measurement

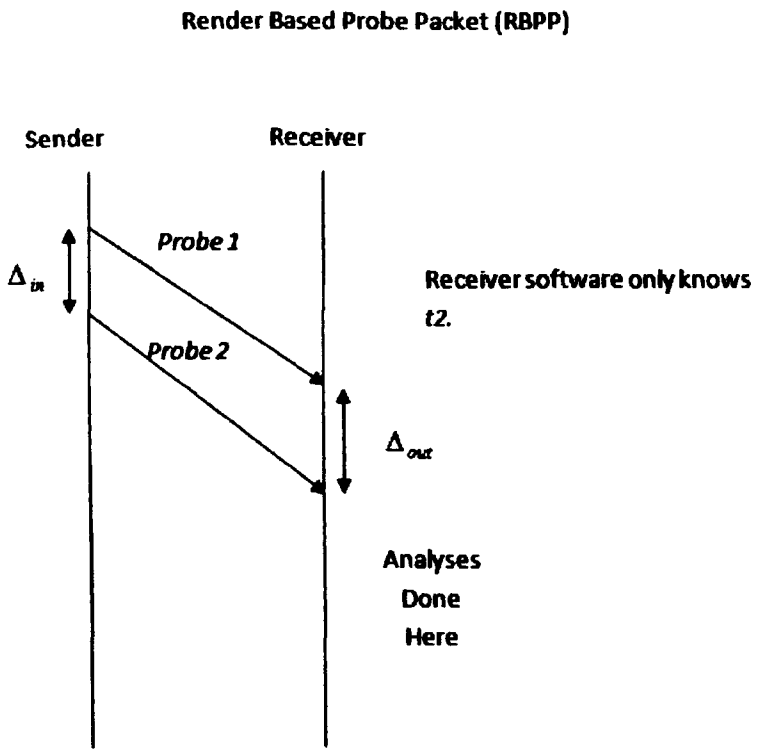


Figure 3.4 Receiver-based packet pair (RBPP) measurement

3.4.3. Packet Pair/Train Dispersion (PPTD) probing

Packet Pair/Train Dispersion (PPTD) attempts to monitor bandwidth non-intrusively by creating short traffic bursts of high bit rate - sometimes higher than the available bandwidth in the path - which last for only a few milliseconds with large silent periods between them. Thus the average probing traffic rate is a small fraction of the available bandwidth. (For instance, the average probing rate in pathload is typically less than 10 percent of the available bandwidth.(Jain and Dovrolis 2004)) Effectively a train of probe packets with a known dispersion (temporal separation) is injected into the monitored path and the dispersion at the receiver end is measured. The narrow link capacity then becomes apparent due to the bottleneck spacing effect, and can be computed using Eqn. (3.2).

However, cross-traffic complicates this simple picture by delaying one or both of the probing packets, thus interfering with the dispersion mechanism. When the first packet is delayed more than the second, the dispersion is increased causing an underestimation of the narrow-link bandwidth. Similarly if the second packet experiences greater delay than the first, the bandwidth is overestimated. If neither packet is delayed by cross traffic (or they both experience identical delay) then the dispersion remains unchanged. This simple picture could be more complicated in a network containing virtual clusters or fabrics, but this lies outside the scope of this work.

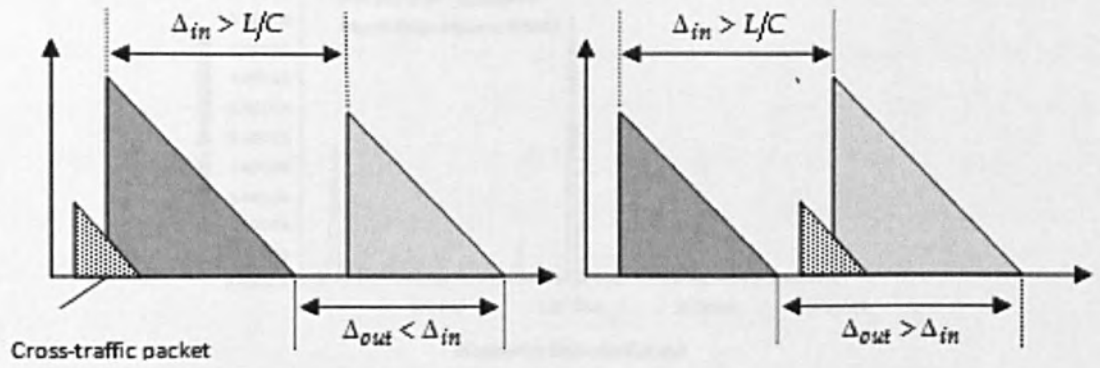
Figure 3.5 illustrates some typical possibilities for packet-pairs. In Figure 3.5(a) there is an idle gap between two packets such that the second packets' behaviour is independent of the first packet. Therefore if neither of the packets is delayed $\Delta_{out} = \Delta_{in}$, and this is referred to as the "independence signature". If the first packet is delayed more than the second then $\Delta_{out} < \Delta_{in}$ and the dispersion is reduced, while if however the second packet is delayed more than the first then $\Delta_{out} > \Delta_{in}$. This produces random noise in the received dispersion, which shall be referred to as "independence noise".

Figure 3.5(b) shows scenarios where there is no idle gap in the link. If no cross traffic packet gets between probe packets then even if the first packet is delayed, $\Delta_{out} = L/C$. This is the "rate signature" already discussed. If cross traffic packets get in between the

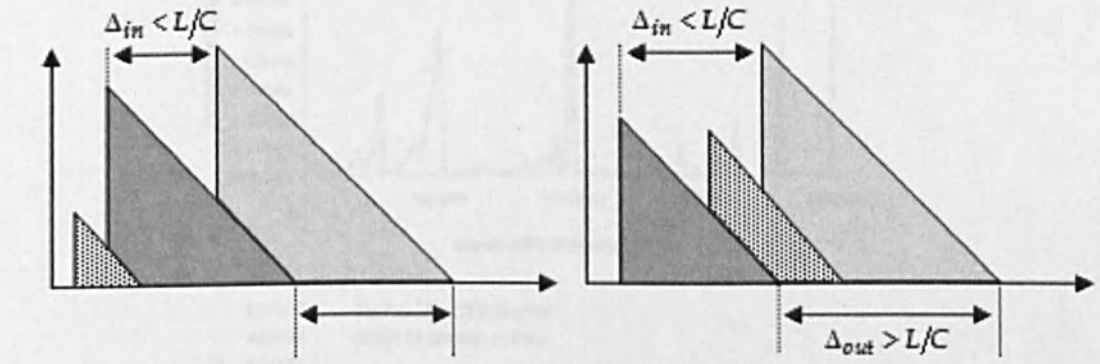
probe packets then $\Delta_{out} > L/C$. The dispersion increase is proportional to the size and number of intervening packets, and is called the "distribute signature" (Tunnicliffe 2010). Thus the "true" bandwidth stands as a local node within the dispersion distribution, surrounded by spurious cross-traffic nodes which must be statistically filtered. Figure 3.5 shows some typical results (Tunnicliffe 2010)for three different probe packet sizes on a network path with a bottleneck rate of 1Mbit/s. Estimates of the bottleneck rate were computed from the raw data using Eqn. (3.1). The bottleneck rate signature (at 1Mbit/s) appears as a local maximum within each distribution. This "true" bandwidth can be identified since it maintains its position as the packet size is changed. When the the packet size is small, the modes are much better defined than when the packet size becomes larger. These results are of course ideal: in reality these nodes may change their positions and sizes as the cross-traffic conditions change dynamically over time(Jain and Dovrolis 2004).

Note that PPTD typically requires double ended measurements, with software running at both the source and the sink of the path (i.e. the receiver-based packet pair (RBPP) approach discussed in Section 3.4.2).

The technique used to study the distributions was usually the histogram method, where measurements are classified into discrete "bins". However Lei and Baker (Lai and Baker 1999) have pointed out some disadvantages, namely that (i) it is difficult to choose a suitable bin width before the distribution itself has been studied, (ii) the possible relationships between data points on opposite sides of a bin boundary are ignored and (iii) lack of relationship between data points on opposite sides of a bin boundary is also ignored. This issue will be addressed further in Chapter 6.



Cross-traffic in a sub-congested link: Output dispersion may be pushed below or above Δ_{in}.



Cross-traffic in a congested link: Dispersion may only be forced *above* L/C.

Figure 3.5 Interactions between probe pair and cross-traffic packets from (Tunnicliffe 2010)

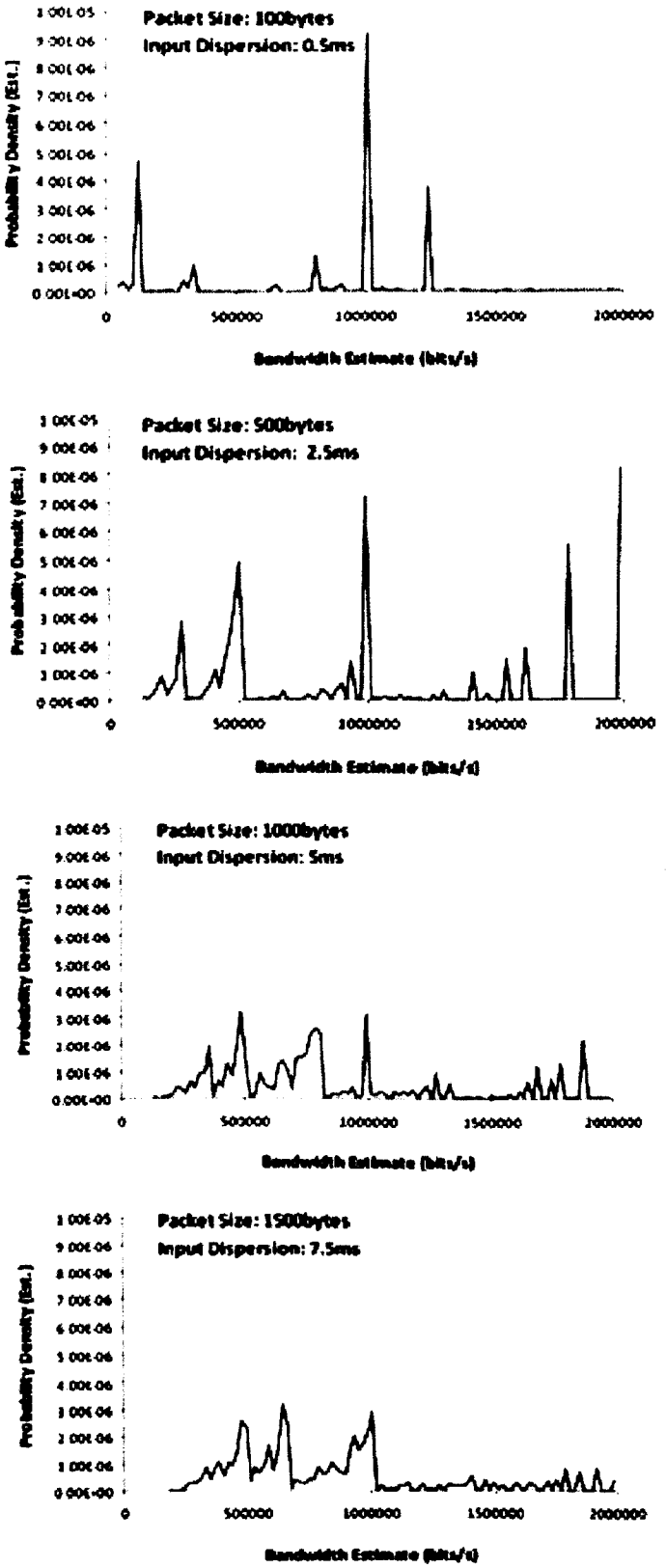


Figure 3.6 Bandwidth Estimation Profiles from (Tunnicliffe 2010)

Another variant of the packet-pair technique have been suggested by Lai and Baker (Lai and Baker 1999). They use a receiver-only method based on existing traffic together with a 'potential bandwidth filtering' scheme. In order to quickly adapt to bandwidth changes they introduce a packet window. When estimating the bandwidth, only packets that arrive within the time interval given by the window will be used. Carter and Crovella (Carter and Crovella 1996) propose two packet-pair based techniques, B-probe and C-probe, to measure bottleneck link bandwidth and available bandwidth, respectively. Paxson (Paxson 1999) has pointed out a number of weaknesses of the packet pair method, such as the problem with multi-channel links, limitations due to clock resolution and out of order packet delivery. To deal with these shortcomings, he introduces an extension of the packet pair technique called the PBM probing technique. There, estimates for a range of probe bunch sizes (i.e. train lengths) are formed and multiple bottleneck values are allowed

3.4.4. Trains of Packet Pairs (TOPP)

The Train of Packet Pairs (TOPP) algorithm is a derivative of PPTD which aims to estimate the *available* bandwidth of the network path(Melander, Bjorkman et al. 2002). (It is sometimes called *DietTopp*, which properly refers to a simplified version devised by Melander et al. in (Melander, Bjorkman et al. 2000)). TOPP has been studied by a number of researchers(Amamra and Hou 2008): The basic idea is similar to SLoPS: TOPP sends many packet pairs with gradually varying dispersion from the source to the sink. If the initial dispersion is Δ_{in} seconds, and the packet-size is L bits, then the *offered rate* of the packet pair $r = L/\Delta_{in}$ bits/s. Similarly the *measured rate* at the receiver is $m = L/\Delta_{out}$ bits/s where Δ_{in} is the dispersion at the receiver end (see Figure 3.7). If r is greater than available bandwidth A then on average $m=r$. If on the other hand, $r < A$ the packet pair arrives at the receiver with the same rate it left at the sender. If only one bottleneck is visible then the governing equation is

$$\frac{r}{m} = \frac{\Delta_{out}}{\Delta_{in}} = \min\left(1, \frac{r}{C} - \left[1 - \frac{A}{C}\right]\right) \tag{3.3}$$

where C is the bottleneck link capacity in bits/s.

TOPP increases the offered rate linearly, while SLoPS uses a binary search to adjust the offered rate. Another important difference between TOPP and SLoPS is that TOPP can also estimate the capacity of the tight link of the path.

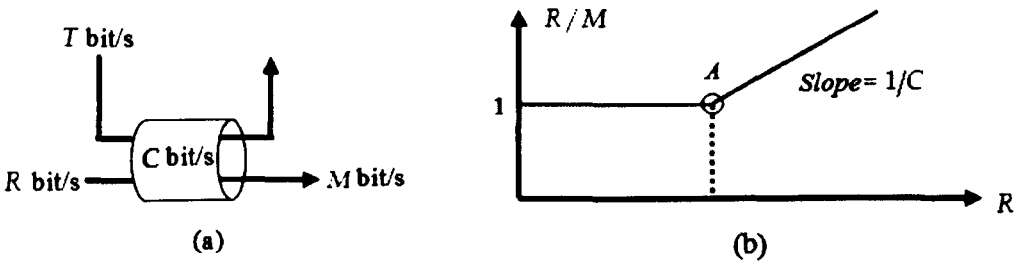


Figure 3.7 The TOPP algorithm. (i) The available capacity is the link capacity l minus the cross traffic c. (ii) An increase in r/m indicates that r>A.

By collecting a set of results for $r > m$, the values of m and l can be computed using least-square analysis. Figure 3.8 shows some typical results for a one-hop network scenario with a bottleneck rate of 2Mbit/s carrying 500 and 5000byte[1] cross traffic at 1Mbit/s (Melander, Bjorkman et al. 2002). As with most studies of this kind, Poisson arrivals are assumed since the superposition of many diverse traffic streams has been found to have this property (Melander, Bjorkman et al. 2002).

The transition between the two linear domains is clearly not abrupt as Eqn. (3.3) would suggest; the dispersion ratios observed in the region $r \approx m$ are considerably higher than their theoretical predictions. This is due to the "probing bias" effect identified by Liu et al(Liu, Ravindran et al. 2004) and discussed in the following section. However, this effect is less pronounced if the probe packets are much larger than the cross-traffic packets (200bytes), and decays rapidly as the offered rate increases. The 5000byte data yields very accurate estimates of the available and link bandwidths (1 and 2Mbit/s respectively).

[1] This experiment assumed a more generic protocol than Ethernet as 10 and 100BaseT prohibit packets larger than 1500 bytes.

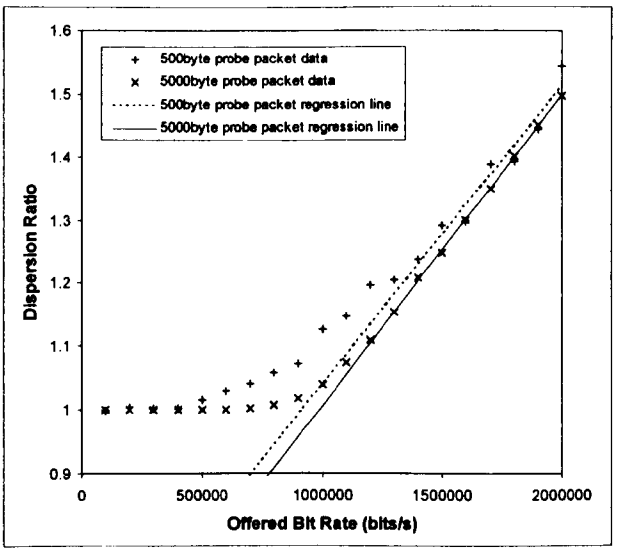


Figure 3.8Typical dispersion results obtained from single-hop simulation using 500 and 1000byte probe packets (Hosseinpour and Tunnicliffe 2007)

The TOPP method can estimate what we call hidden bottlenecks (when the surplus bottleneck is not on the link bottleneck). The TOPP regression analysis can not only estimate the surplus bottleneck and link bandwidth but also the surplus and link bandwidths of several hops under some assumptions. TOPP is network friendly. It uses trains of packet pairs and the distance between pairs is variable in order to avoid overloading the network with bursts of packets and to minimize the impact on cross traffic.

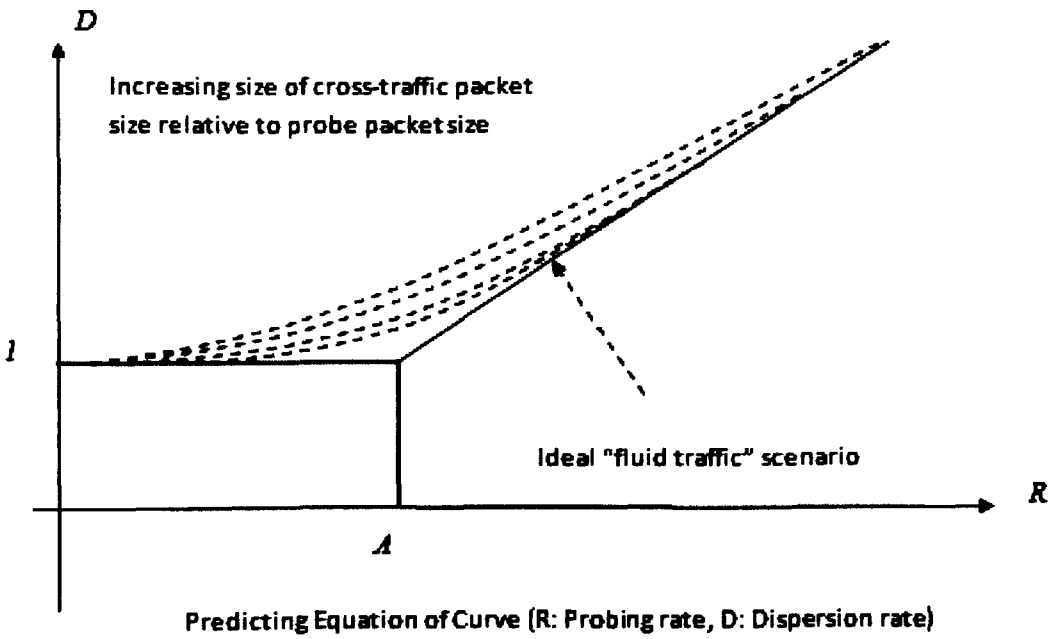


Figure 3.9 Effect of packet size on TOPP graphs. (The dispersion ratio $D=R/M$.)

3.4.5. Probing Bias

The probing bias previously mentioned is an important phenomenon as it is potentially responsible for causing inaccuracies in available bandwidth estimation in TOPP. The probing bias has been shown to decrease when the probe packet size increases relative to the cross-traffic packet size, and as the number of packets in the train increases (Tunnicliffe 2010). However, since it is beneficial to keep probe-packet sizes and train lengths small to reduce probing overhead, several attempts have been made to model probing bias such that it can be included in calculations.

One of the earliest studies was published in 2005 by (Liu, Ravindran et al. 2005) who used a deterministic sample-path approach, while the following year (Park, Lim et al. 2006) approached the same problem from a probabilistic angle. Using the M/D/c queuing model of (Franx 2002) they established a relationship between the input and the output probing gaps in the single-hop and multi-hop cases under the assumption of stationary Poisson cross-traffic, and showed that the proposed model agrees very well with

simulations. They further showed that simpler models previously published can be regarded as special cases of this general model. However, the complete general model was rather complex and unwieldy, assumes the cross-traffic packets are all of equal size, and that the probe packets are equivalent to a whole number of cross-traffic packets.

In 2007 (Ha´ga, Diriczi et al. 2007) approached the same problem using the Takacs integro-differential equation for a queuing system, producing a formula very similar to that of (Liu, Ravindran et al. 2004) which was again based on Poisson traffic. Haga's model included a "granularity factor" allowing a mix of different packet sizes in the cross-traffic. However, the model still requires numerical solution, and is therefore computationally costly. A third solution was offered in 2008 by (Tunnicliffe and Winnett 2009), who used a dynamically evolving Gaussian function to represent the queue-size distribution between the probe packets. Unlike the Park and Haga models this model was approximate, but was shown to be accurate for utilisations up to about 70%, and when the probe packets exceeded twice the size of the cross-traffic packets. This model was later improved to include an "effective packet size" (similar to Haga's "granularity factor") to represent non-constant packet sizes distributions (Ha´ga, Diriczi et al. 2007).

3.5. Software Tools for Probing

Several specific software tools have been developed as implementations of the above-described techniques, sometimes with subtle modifications. A few examples are listed below:

3.5.1. Pathchirp

The pathChirp tool uses self-induced congestion from a "chirp train" of probe packets to estimate the available bandwidth. Like TOPP, PPSD and SLoPS its uses a short term streams to create and detect temporary congestion while not disrupting the overall flow of traffic. Like TOPP and PPSD (and unlike SLoPS) it uses inter-packet gaps rather than end-to-end delays, so as not to need synchronisation. It also obtains more information from fewer packets, by means of decreasing in to the inter-packet gaps within a chirp, and therefore imposes less overhead than other packet-train methods (Ribeiro, Riedi et al. 2003). However, it has been noted that there are many scenarios where pathChirp cannot provide an accurate measure of available bandwidth.

3.5.2. Spruce

Spruce (Suthaharan and Kumar 2008) (Spread PaiR Unused Capacity Estimate) samples the arrival rate at the bottleneck by sending pairs of packets spaced so that the second probe packet arrives at a bottleneck queue as the first packet departs and thus calculates the number of bytes that arrived between the two probes from the inter-probe spacing at the receiver. Spruce then computes the available bandwidth as the difference between the path capacity and the arrival rate at the bottleneck. Spruce differs from TOPP in that (i) it needs to know the value of the physical bandwidth (link rate *C*) and that (ii) it uses a single value of Δ_{in} to measure a mean value of Δ_{out} and compute $A = C\left(1 - \frac{\Delta_{out}-\Delta_{in}}{\Delta_{in}}\right)$. (TOPP meanwhile uses many values of Δ_{in} to create different ralues of $r = \frac{l}{\Delta_{in}}$ and applies regression analysis to the resulting graph.)(Strauss, Katabi et al. 2003).

3.5.3. Abing

Abing also derives estimates of available bandwidth from the delay introduced between paired packets. It is based on the same principle as Spruce, so it has similar advantages and disadvantages relative to TOPP and PPSD. However, Spruce uses Poisson process to space the packet-pairs, thus obtaining more accurate averages due to the PASTA (Poisson Arrivals See Time Averages) property, while Abing periodically injects its packets in to the link.(Shriram, Murray et al. 2005).

3.5.4. IGI/PTR

Gap increasing (IGI) and packet transmission rate (PTR) are two more packet-pair techniques for characterizing the available bandwidth on a network path. The techniques experimentally determine an initial packet-pair gap that will yield a high correlation between the competing traffic throughput on the bottleneck link and the packet gap at the destination. IGI is like TOPP but uses 60 packet streams. PTR is very similar to Spruce but also uses 60 packet streams.(Hu and Steenkiste 2003).

3.5.5. Pathload,

Pathload is a specific implementation of SLoPS. Tests have shown it to be nonintrusive and that does not cause significant increases in network utilization, delays, or losses. However, unlike dispersion methods it needs synchronization between sender and receiver clocks.(Jain and Dovrolis 2002)

3.6. Application to Wireless Technologies

The extension of probe-based measurement to wireless and wired-cum-wireless network environments has been problematic, due to such factors as dynamic rate-switching implemented in Wi-Fi technologies and fading channels which may switch abruptly to a lower data-rate, confusing the probe software (Lakshminarayanan, Padmanabhan et al. 2004). Pathchirp has been shown to produce superior results in mobile networks(Castellanos, Villa et al. 2006). Also Sarr et al.(Sarr, Chaudet et al. 2008) showed how the nodes in an ad hoc network can estimate channel occupancy non-invasively, without the need for probe packets. They have also found that available bandwidth in wireless networks undergoes fast time-scale variation because of channel fading and physical obstacle errors(Shah, Chen et al. 2003). Also since a wireless channel is a shared-access medium, available bandwidth varies with the number of hosts using the same channel.

3.6.1. Wired-cum-Wireless and Access Networks

Several workers have investigated bandwidth probing in wired-cum-wireless networks over the past decade. One of the earliest studies was by Mascole et al. (Mascolo and Vacirca 2005) who proposed and investigated "TCP Westwood", a TCP extension for wireless based on available bandwidth measurement by monitoring ACK packets at the sender. The available bandwidth estimate was shown to respond accurately to step changes in UDP traffic on the same channel. This was extended further in 2004 by Nadim Parvez and Ekram Hossein (Parvez and Hossain 2004) who exploited the burstiness pattern of ACK arrivals to produce more accurate bandwidth measurements. (This lat scheme was called "TCP Prairie".) In 2004 Lakshimarayanan et al. (Lakshminarayanan, Padmanabhan et al. 2004) investigated wireless broadband access networks with token

bucket rate regulation and non-FIFO queuing. Both of which are problematic in conventional probing. Their "PROBGAP" method was to send Poisson-spaced probe packets each with a 20-bytes payload and a time stamp, and based its measurement on one-way delay. The "knee" point of the cumulative delay profile was used as a measure of idle rate (the fraction of time the link is idle) which is multiplied by the link rate to yield available bandwidth. This was found to work well when the broadband ink was considered in isolation, but more problematic in wider setting. In 2005 Johnsson (Johnsson, Melander et al. 2006) at el. applied DietTOPP to a wired-cum-wireless connection and discovered that the measured available bandwidth and link capacity depends on the probe packet size as well as the cross-traffic intensity. DietTOPP estimates were found to decrease with increasing probe packet size, which was attributed to the large relative overhead imposed by the 802.11 MAC(Johnsson, Melander et al. 2006). Also in 2005 (Shriram, Murray et al. 2005) compared the performance of several probing techniques including Abing, Pathchirp and Spruce for estimating bandwidth on a wired-cum-wireless testbed. They found that Pathload and Pathchirp were the most accurate, while Abing and Spruce were vulnerable to delay quantization. In 2008 (Li, Claypool et al. 2008) combined the packet-pair and train approaches to create a new technique WBest, which avoided the use of search algorithms to pimpoint the bandwidth. When compared with Pathchirp, Pathlod and IGR/PTR, this method proved faster, more accurate and less intrusive. In the same year, Michael Bredel and Markus Fidler (Bredel and Fidler 2008) compared different techniques on wireless, confirming that the FIFO assumption is not valid on wireless channels with contention. Their results suggested that techniques specifically aimed at wireless channel measurement performed no better than the techniques developed for wired networks.

3.6.2. Wireless Mesh and Ad-Hoc Networks

Much of the work on wireless mesh and ad-hoc networks has been based upon "passive methods" whereby available bandwidth is computed from the link "idle rate" (the proportion of time the link is idle). The idea is similar to the PROBEGAP method mentioned earlier, but dispenses entirely with probe packets. One example id the "IdleGap" method proposed by (Lee, Hall et al. 2007) in 2007, in a study of bandwidth measurement in wireless networks carrying multimedia streaming packets. IdleGap

efficiently calculated the available bandwidth by collecting the information of one node to determine the idle rate, which when multiplied by the link capacity gave a measure of available bandwidth. In the same year (Kliazovich and Granelli 2006) presented a cross-layer congestion avoidance scheme called "C^3TCP" whose bandwidth estimation mechanism used timestamp information for individual packets. However, this required the implementation of new link-layer modules in the protocol stack. In 2008 (Amamra and Hou 2008) proposed a new probe-based technique called "SLOT", combining the TOPP and SLoPS mechanisms, specifically aimed at measuring bandwidth in an Ad-hoc wireless sensor network. It was shown to have a shorter probing time than TOPP and to provide more accurate bandwidth estimation than SLoPS. In the same year (Sarr, Chaudet et al. 2008) presented a new passive technique to measure the available bandwidth on the IEEE 802.11 MAC layer using *hello* packets to carry idle rate information between nodes. They measured available bandwidth between two neighbor nods as the maximum throughput, and noticed that the passive method worked better in this environment than active (probe based) methods. The same conclusion was reached by (Gupta, Wu et al. 2009) in 2009, when they experimentally compared the passive and active methods on wireless networks. On the basis of their results they proposed that the probe-based tools are not the best choice for wireless networks.

3.7. Summery and Conclusions

This chapter compares the different bandwidth measurement techniques and looks at the different methods of packet pair measurements in wired and wireless networks. All the software tools for bandwidth probing can be summarized as follows:

- Intermediate Device-Dependent Methods
    - Variable Packet Size (VPS) probing
    - SNMP-Based Measurements
- Methods Relying on One-Way Latencies
    - End-to-End Dispersion-Based Methods
    - Bottleneck Spacing Effect
- Sender and Receiver-Based Methods
    - Packet Pair/Train Dispersion (PPTD) probing

- Trains of Packet Pairs (TOPP)

The main reason for concentrating on end-to-end dispersion approaches is that it might not be possible to access to other nodes in the path (since not all devices implement TCP/IP or allow administrative access to their SNMP agents) and it is therefore best to make measurements from one end to the other.

4. DESIGN AND VALIDATION OF OPNET SIMULATION TESTBED

4.1. Introduction

For practical reasons this research is based on simulation rather than on hardware experimentation. It was therefore necessary to choose or develop a simulation software which could demonstrably mimic the behaviour of real network components. A variety of simulation tools were considered, including the following:

- OPNET is a proprietary tool for network modelling and development, designed by OPNET Technologies, Inc. (OPNET Technologies 2010) and commonly used by network consultants and other professionals in the industry. The basic package is called Modeler, which provides a flexible and scalable environment for the design and study of communication networks, through graphical models and C++ programming. The models are divided into three layers: the Process model, the Node Model and the Project model. By creating simple Process and Node models, a large scale Project model may be built with few adjustments. Meanwhile the object-oriented C++ is used to program the functionality of the models at the bottom level. When a model has been implemented, its parameters can be defined and its behaviour monitored during simulation. Tools for the analysis, evaluation and comparison of results are also available. The ability to simulate wireless systems requires the add-on Wireless Module which includes the IEEE 802.15.4/ZigBee and IEEE 802.11g/Wi-Fi standards.
- NS2 (version 2) is an object-oriented discrete event driven network simulator developed at UC Berkely which is primarily useful for simulating local and wide area networks. Although NS2 is easy for the experienced user, it is more difficult for a first time user because there are few user-friendly manuals. (Although there is a lot of in-depth documentation written by the developers, it is written mainly for the skilled NS2 user.)

- **OMNeT++ is** another discrete event simulation environment for communication networks (though because of its generic and flexible architecture, it is also successfully used in other areas, such as IT systems, queuing networks and hardware architectures. Its is based on a component architecture, the basic components programmed in C++ and assembled into larger components and models.

OPNET was chosen for the early part of this study for a number of reasons. Firstly it is an industry standard tool which can be trusted to produce a reliable prediction of real network behaviour. In addition, OPNET is distinguished from other similar tools by its flexibility and ease of use. Unlike many other tools, it can incorporate new communication network technologies and its hierarchical structure allows very large and complex networks to be studied. It also has an integrated development environment to help model and evaluate communication networks and distributed systems. Furthermore, though OPNET is a proprietary tool, it is installed and available within the Faculty and could therefore be used without incurring any additional charges.

This chapter reports how an OPNET-based simulation testbed was established to investigate the packet-pair bandwidth measurement mechanism in wired, wireless and wired-cum-wireless network environments, allowing the monitoring techniques described in the previous chapter to be implemented and explored.

4.2. Overview of Opnet Architecture

The OPNET package comprises a number of useful tools, each of which focuses on some particular aspect of modelling. They fall into three major categories which correspond to the three phases of modelling and simulation, namely (i) *Specification,* (ii) *Data Collection and Simulation* and (iii) *Analysis*. Figure 4.1 shows the simulation project cycle in OPNET simulation tool (OPNET Technologies).

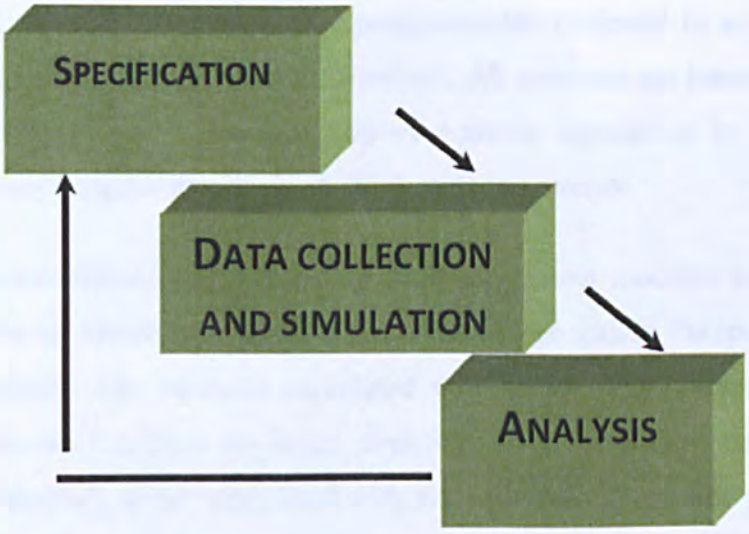


Figure 4.1 Simulation Project Cycle

In order to provide useful data, network models have to combine accurate descriptions of topology, data flow and control flow. Since no single paradigm of visual representation is ideally suited for all three, OPNET utilizes three separate model formats, which fit together in a hierarchical fashion:

4.2.1. Network Modelling Domain

The Network Model defines the position and interconnection of communicating entities or "Nodes", which include host computers (workstations and servers), routers and switches, via a variety of different link technologies. The structure is hierarchical, allowing smaller networks to be defined as "subnets" (not to be confused with IP subnets) which may be duplicated and interconnected at higher levels within the hierarchy. Components and links may be selected pre-configured from a "palette", or alternatively custom-constructed to meet the user's requirements

4.2.2. The Node Modelling Domain

The OPNET Node Editor allows communication devices to be created and edited in terms a block structured data flow diagram. The functionality of each programmable block in a node is defined by a Process Model (see below). There are several different kinds: Some have predefined characteristics and parameters (such as point-to-point transmitters and

radio receivers) while others are highly programmable (referred to as processors and queues) and rely on process model specifications. All modules are interconnected either by "statwires" (statistic wires, used to convey numeric signals) or by packet streams. Statwires commonly trigger interrupts on the destination module.

Packet streams are objects in Node Editor used to connect modules together, and the streams connecting source and destination modules are called "output" and "input" streams respectively. The methods associated with input streams (streams entering a destination processes) include op_intrpt_stream(), op_pk_get() and op_strm_pksize(). Output stream methods (those associated with streams leaving a sender process) include Op_pk_send(), op_pk_send_delayed() and op_pk_send_forced().

Packets are the information-carrying entities that circulate among system components. Packet streams flow between modules and there is a built-in packet buffer at the end of every stream. They are dynamic simulation entities which are continuously created and destroyed as the simulation progresses. A single system could use multiple types of packets with different formats. There are three methods for transferring a packet and notifying arrivals: firstly scheduled arrivals which occur after all other events, secondly forced arrivals which will occur immediately, and thirdly quiet arrivals are those for which no stream interrupt occurs.

4.2.3. The Process Modelling Domain

The Process Editor is used to create and edit the processes internal to entities in the Node Domain, thus describing the logical flow and behaviour of the processor and queue modules. Process model executives are expressed using a combination of tools: programming languages (C++ and Proto-C), State Transition Diagrams (STDs) and a library of "kernel procedures". In principle the STD approach may support any type of protocol, resource, application, algorithm, or queuing policy. State transitions occur in response to interrupts (see Figure 2) which support communication between processes. States and transitions graphically define the progression of a process, while the general logic is specified using a library of predefined functions and the flexibility of the C language. Processes may also create new "child" processes to perform sub-tasks. Table 4.1 describes components of STD [OPNET_STD].

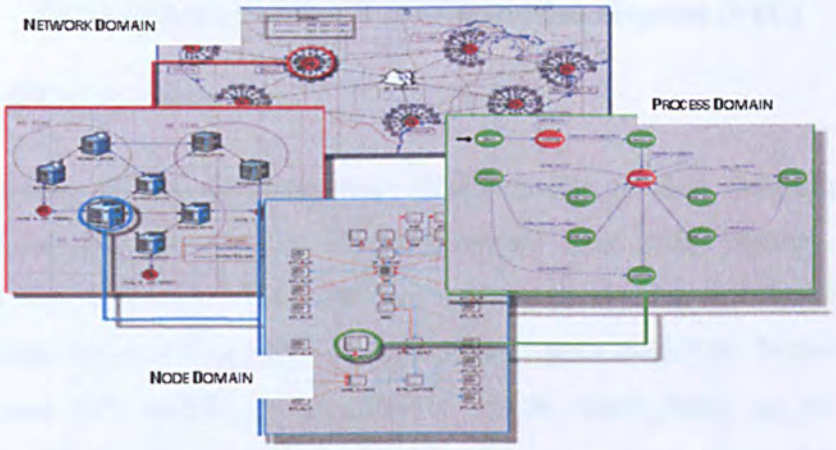


Figure 4.2 OPNET Modelling Hierarchy (OPNET Technologies)

STD component	DESCRIPTION
State Variables	Processes maintain private state variables with named variables of arbitrary data types including OPNET-specific, general C/C++ language, and user-defined types. This capability allows the process to maintain counters, routing tables, statistics related to its performance, or messages requiring retransmission. Arbitrary combinations of state variable values may be used in all decisions and actions implemented by a process.
State Executives	Each state of a process can specify arbitrarily complex actions associated with the process entering or leaving that state. These actions, called *state executives*, are expressed in C/C++ language. Typical actions include modifying state information, creating, receiving, updating and sending messages, updating statistics, and setting or responding to timers.
Transitions Conditions	These determine whether a state transition should occur, and are expressed in C/C++ language. They can make reference to properties of a new interrupt as well as to combinations of state variables.
Transition Executive	Transition may specify general actions called *executives* that are implemented each time that they are traversed.

Table 4.1 Components of state transition diagram (STD)

4.2.4. Kernel Procedures

Kernel Procedures (KP's) are pre-written functions that abstract difficult, tedious, or common operations, thus freeing the programmer from such matters as memory management, data structures and the handling of event processing. All Kernel Procedures begin with prefix op_ and they all focus on communication modelling. Some examples of commonly used KPs include op_pk_create(), op_pk_create_fmt(), op_pk_copy() and op_pk_get()In KPs, all the variables, regardless of their category, have some properties like a name, a datatype and a value. The name and datatype of a variable remain fixed throughout the life of a process. The value component refers to the actual contents of memory that is associated with a variable. For simple variables this may be an individual number or string of characters; for compound data structures many such items could be involved. Table 4.2 shows the essential OPNET KPs predefined data structures used in process model. [OPNET_ Process]

Data Structure	Description	Data Type
Compare codes	These indicate whether an operation completed successfully.	`Compcode`
Distributions	These describe the mapping of a random number to a specific numeric outcome consistent with a probability density function (PDF)	`Distribution`
Event Handles	These identify a pending simulation event (interrupt).	`Evhandle`
Statistic handles	These identity the global and local statistics that are created dynamically.	`Stathandle`
Interface Control Information	These are collections of structured data that are used to associate additional user	`Ici`

	defined information to an event.	
Lists	These are ordered collections of data elements, which can range from simple C/C++ data types to complex data structures.	`List`
Object ID	This is a numeric serial number which uniquely identifies a single simulation object.	`Objid`
Packets	These are structured groups of data fields representing messages that can flow over packet streams in the node domain, and over links in the network domain.	`Packet`
Log handles	Used when creating simulation logs to aid simulation debugging or results analysis.	`Log_handle`
Process Handles	These are used to identify an active process within a simulation.	`Prohandle`

Table 4.2 OPNET Kernel procedures

Categories	Description
State variables (SV)	These are static variables in a process that are visible to all states. (They are equivalent to global variables in C programming.)
Temporary variables (TV)	These variables are only visible within a state; they are no longer meaningful once the process exits that state.
Header block (HB)	This is similar to a header file in C programming, and contain related include files and macros, as well as the state transition conditions. (In OPNET STD conditions

	are defined as macros.)
Function Block (FB)	This stores common produces and help functions invoked by programs in enter-exec or exit-exec.
Diagnostic Block (DB)	OPNET simulations automatically incorporate the OPNET Diagnostic Block (ODB) that allows the user to interactively monitor the progress of a simulation and the state of its objects.
Termination block (TB)	Defines procedures to be invoked when the simulation terminates.

Table 4.3 Proto-C categories

4.3. Packet-Pair Generation Mechanism in OPNET

Now that the general characteristics of OPNET have been described, this section shows how the architecture was employed to generate packet pairs. Figure 4.3 shows an FSM designed early in the project, illustrating the different states and state transitions. The red circles represent "unforced" states, while the green circle represents a "forced" state. When the probing process begins, the machine moves from its initial state to the "wait" state. The arrival of a packet forces it to shift to the "send data" state, from which it naturally returns to the "wait" state until the next packet arrives. The FSM also contains programming code, specified in the enter and exit executives of the states, and in header and function blocks associated with the module as a whole. This section describes the design and implementation of the packet-pair generation mechanism as an OPNET module.

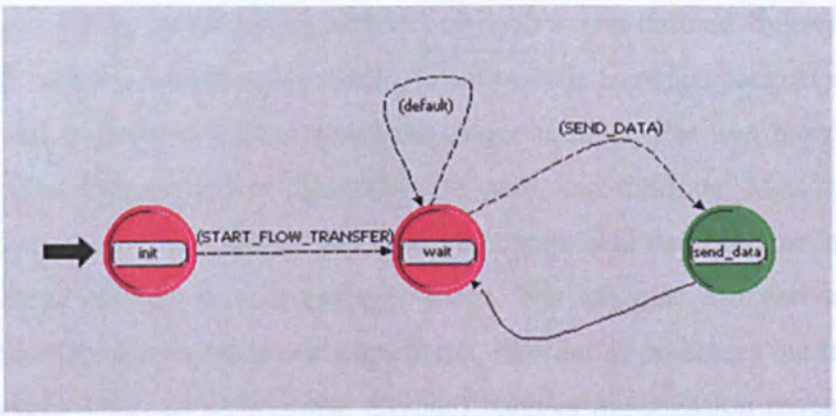


Figure 4.3 OPNET Finite State Machine (FSM) for packet-pair generation

4.3.1. Advanced Ethernet Workstation Model – General Overview

The device sending packet pairs was based upon the advanced Ethernet workstation model (ethernet_station_adv) available from the Opnet node palette. Figure 4.4 shows the node model; the protocol stack is relatively simple and includes only 3 layers: an Application Layer, a Link Layer and a Physical Layer. There are no IP and Transport Layers, since the node simply sends and receives Ethernet frames generated according to a model specified in the bursty_gen module.

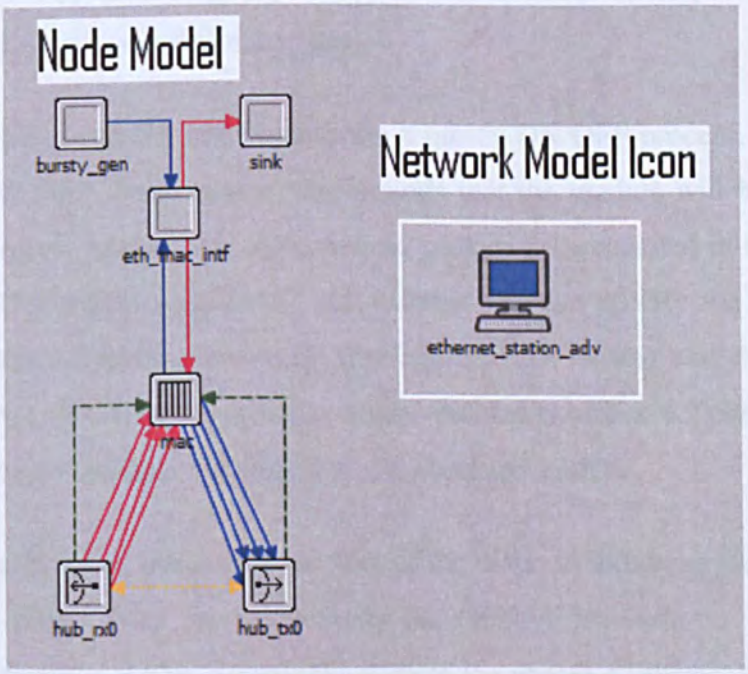


Figure 4.4 OPNET Ethernet_work_station_adv

This bursty_gen model generates packets according to a user-defined "bursty" (ON/OFF) model, which switches between two states: An active state in which packets are generated periodically and an inactive state in which the source is silent. The user has the facility to define the packet format, packet size, data rate, start/stop time and MAC address. The sink module calculates the transfer time (and other statistical data) for incoming packets and deletes them in order to free memory space. The eth_mac_intf and mac modules implement the Ethernet protocols and algorithms, in order to processes the incoming and outgoing packets. Ethernet MAC (mac module) handles transmission requests from the higher layers, encapsulates packets into Ethernet frames and sends these frames to the other side of the interface. The speed at which the MAC operates is governed by the data rate of the connected link. MAC can be configured to operate in half duplex, but here it is used in full duplex.

The attributes of ethernet_station_adv node model is shown in Figure 4.4. "Highest Destination Address" and "Lowest Destination Address" specify the upper bound and the lower bound on a destination addresses to be chosen at random. Here since the SV is broadcasted in the network, we set this attribute "broadcast" to ensure that all receiving MACs will accept the frame, regardless of their own address. In "Traffic Generation Parameters" column, one can specify the parameters of the traffic pattern that will be generated by this traffic source (bursty_gen).

As mentioned before, bursty_gen implements a bursty ON/OFF process. The "ON State Time" and "OFF State Time" specify the periods that the module will be in each state. Packets are generated in the "ON" state, and no packets are generated in the "OFF" state. In the "Packet Generation Arguments" sub column one can specify the parameters that determine the rate of packet generation (during the "ON" state) and the size of these generated packets. OPNET also provides a user-defined statistics collection function: In this work, the end-to-end delay is chosen as the observed statistic.

The hub_rx0 and hub_tx0 modules at the foot of the diagram represent the Physical Layer of the protocol stack. They are respectively the OPNET symbols for a point-to-point receiver and transmitter, which connect the node to the physical Ethernet connection.

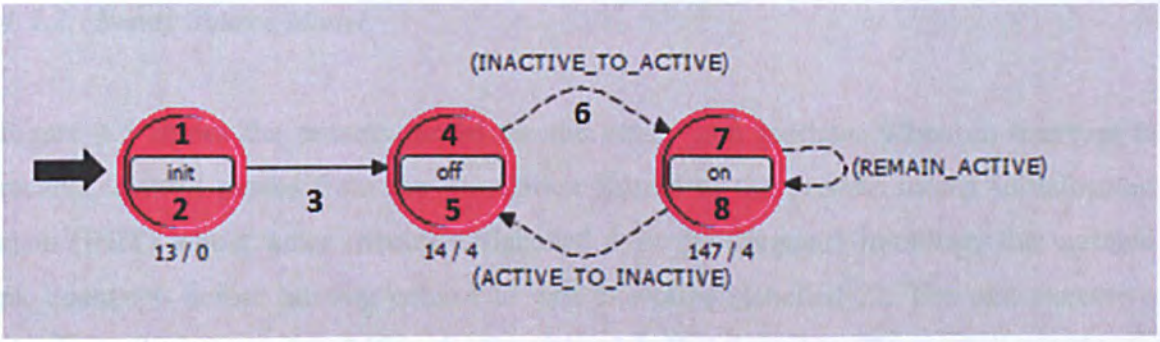


Figure 4.5 Process model for bursty_gen module

BURSTY SOURCE MODULE (Figure 4.5)

1. Initial interrupt delivery. Enter exec invoked which initializes PK_count==0

2. Exit exe invoked immediately. Transition condition (pk_count==0) evaluated true.

3. Transition occurs since pk_count==0

4. OFF enter executive invoked

5. After off_time has elapsed, the OFF state's exit exec is invoked. Transition condition INACTIVE_TO_ACTIVE evalutes to true

6. Transition occurs INACTIVE_TO_ACTIVE /ACTIVE_TO_INACTIVE

7. Enter exec will create packet and will remain creating new packets until transition condition REMAIN_ACTIVE evaluates false

8. Exit exec will send every packet to the next module. After ON period the transiton condition (ACTIVE_TO_INACTIVE) evaluates to true.

SINK MODULE (Figure 4.6)

1. INITDISCARD1. Initial interrupt delivered and the enter exec invoked which increment the pk_count

2. Exit execs will write the pk_count value in record statistic and pass the control to discard state

3. Exit exec will delete the arrived packet and pass the control back to module2.

Table 4.4 Summary of bursty_gen and sink module operations

4.3.2. Bursty Source Model

Figure 4.5 shows the process model for the bursty_gen module. When an interrupt is receive, control passes from the Simulation Kernel to the process model initialization state (INIT) whose enter executive (labelled 1 in the diagram) initializes the variable pk_count==0 before passing control to exit executive (labelled 2). The exit executive checks if pk_count==0 and if so the transition labelled 3 occurs. The control remains in the OFF state until the transition condition INACTIVE_TO_ACTIVE becomes true. When the control transferred to ON state, this creates new packet. Control is transferred back to the OFF state when the transition condition ACTIVE_TO_INACTIVE evaluates true. The process is summarised in Table 4.4.

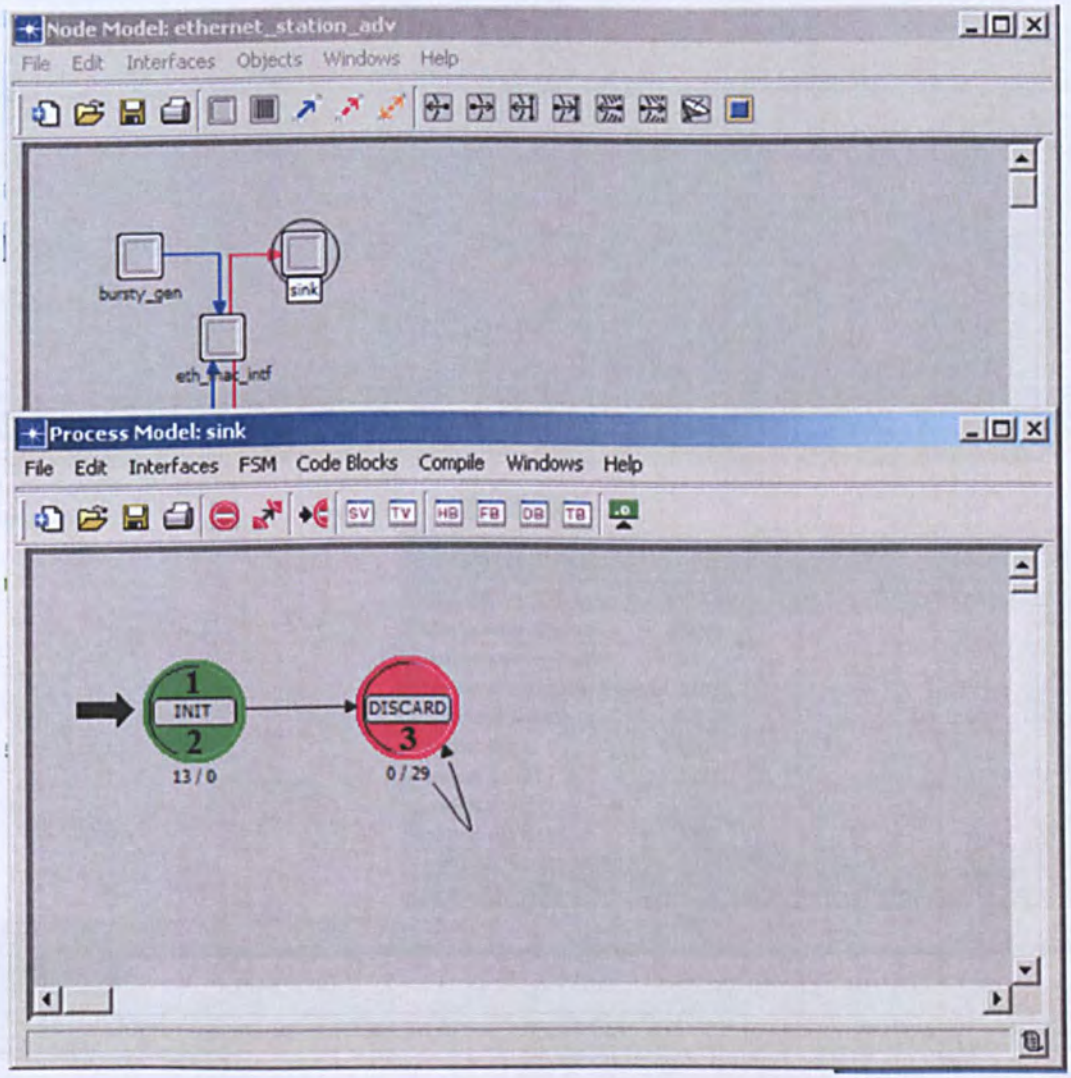


Figure 4.6 Process model for sink module

4.3.3. Sink Model

Figure 4.6 shows the process model for the sink module. When an interrupt is delivered to the module, control passes from the Simulation Kernel to the process model initialization state (INIT). The INIT state increments the packet count variable and passes the control to its exit executive. The exit executive writes the value of the packet count in record statistics using the script op_stat_write(pk_cnt_stathandle, pk_count) and passes control to DISCARD state. The DISCARD state deletes the packet and passes control back to the SINK module, which will pass the control back to simulation kernel. The process is summarised in Table 4.4.

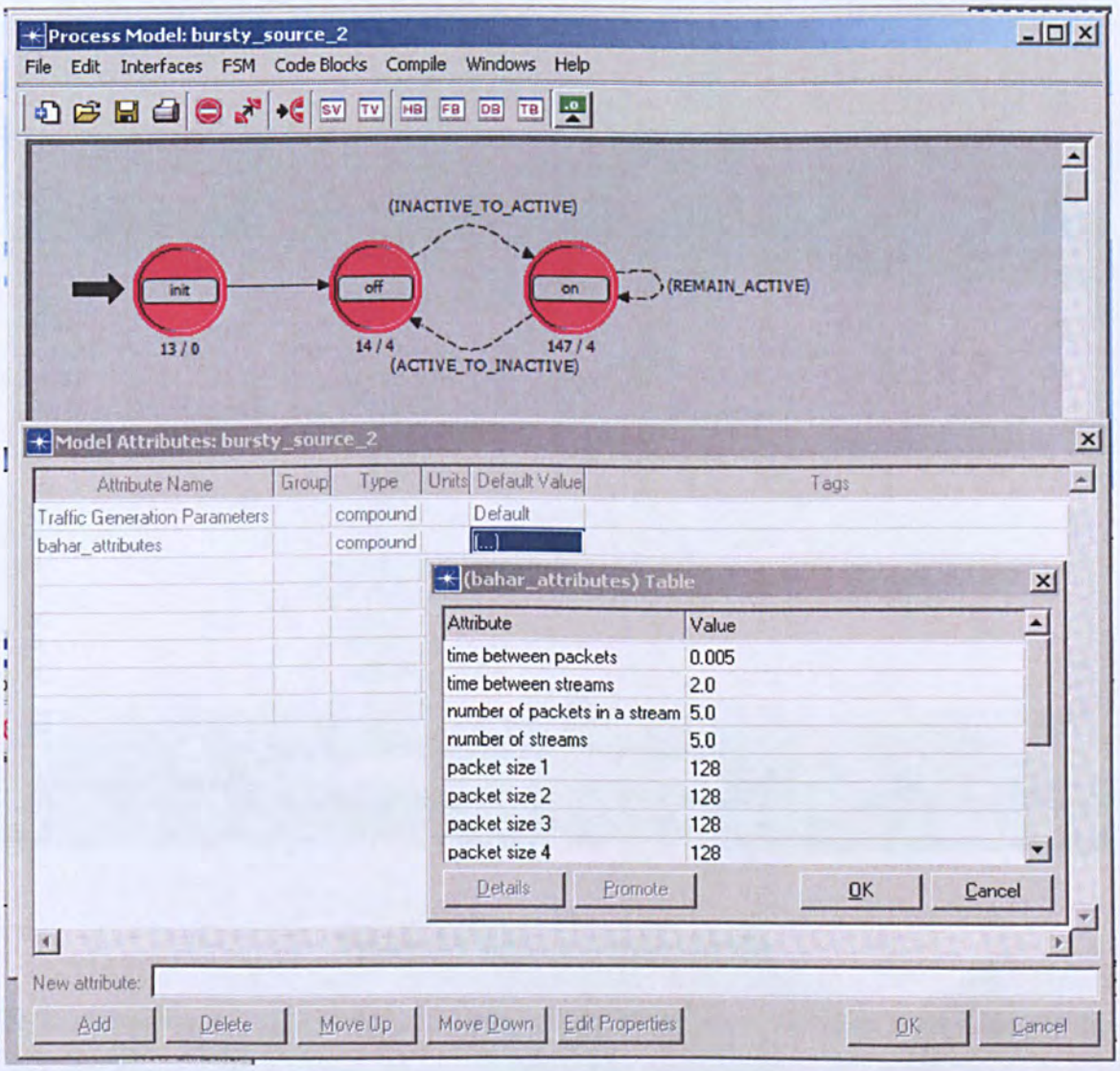


Figure 4.7 Attributes dialogue for modified bursty source

4.3.4. Modified Version to Generate Packet Pairs/Streams

The workstation model with the bursty source was modified to generate pairs and streams of packets of the sort necessary for the probing algorithms described in Chapter 3. Figure 4.7 shows the attributes added in the modified version: Streams can consist of up to ten packets, and each packet can if necessary be given a different size. The times between packets, times between streams, numbers of packets in a stream and mumber of streams can all be set by the user via this dialogue.

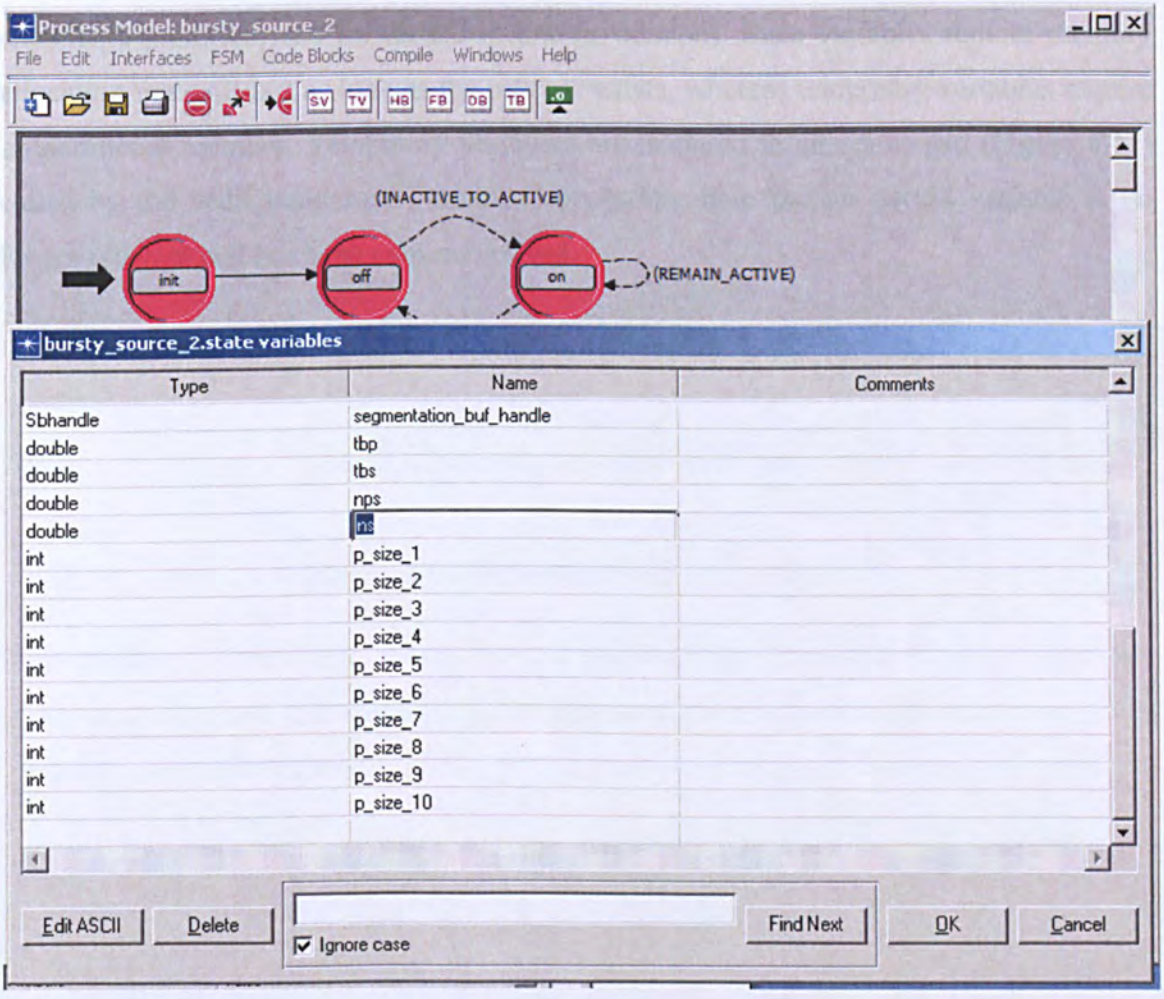


Figure 4.8 Editing state variables

To implement this functionality, a number of additional state variables were added to the module. (State variables are those variables that retain their value from one process invocation to the next, unlike temporary variables which retain their values only during

the span of a single process model invocation). The "add state variable" operation (see Figure 4.8) allows these variables to be added and deleted using the "Edit ASCII" button. The new variables added are tbp (time between packets), tbs (time between streams), nps (number of packets per stream), ns (number of streams) and p_size_1....p_size_10 (the sizes of the packets in bytes). Temporary variables store information that does not require persistence, and are not guaranteed any particular set of values when a process model is invoked; values assigned by the process during previous invocations may have changed. They therefore provide "scratch" storage of information relevant only within the lifetime of a process invocation (e.g. the index variable of a loop, or a packet that is received, examined, modified, and forwarded in one invocation). State variables require memory allocation that will last as long as the process exists, whereas temporary variables require no additional memory. Temporary variables are declared in an editor pad (Figure 4.11) called by the "edit temporary variables" operation; here the on_period variable is no longer relevant and has been commented out.

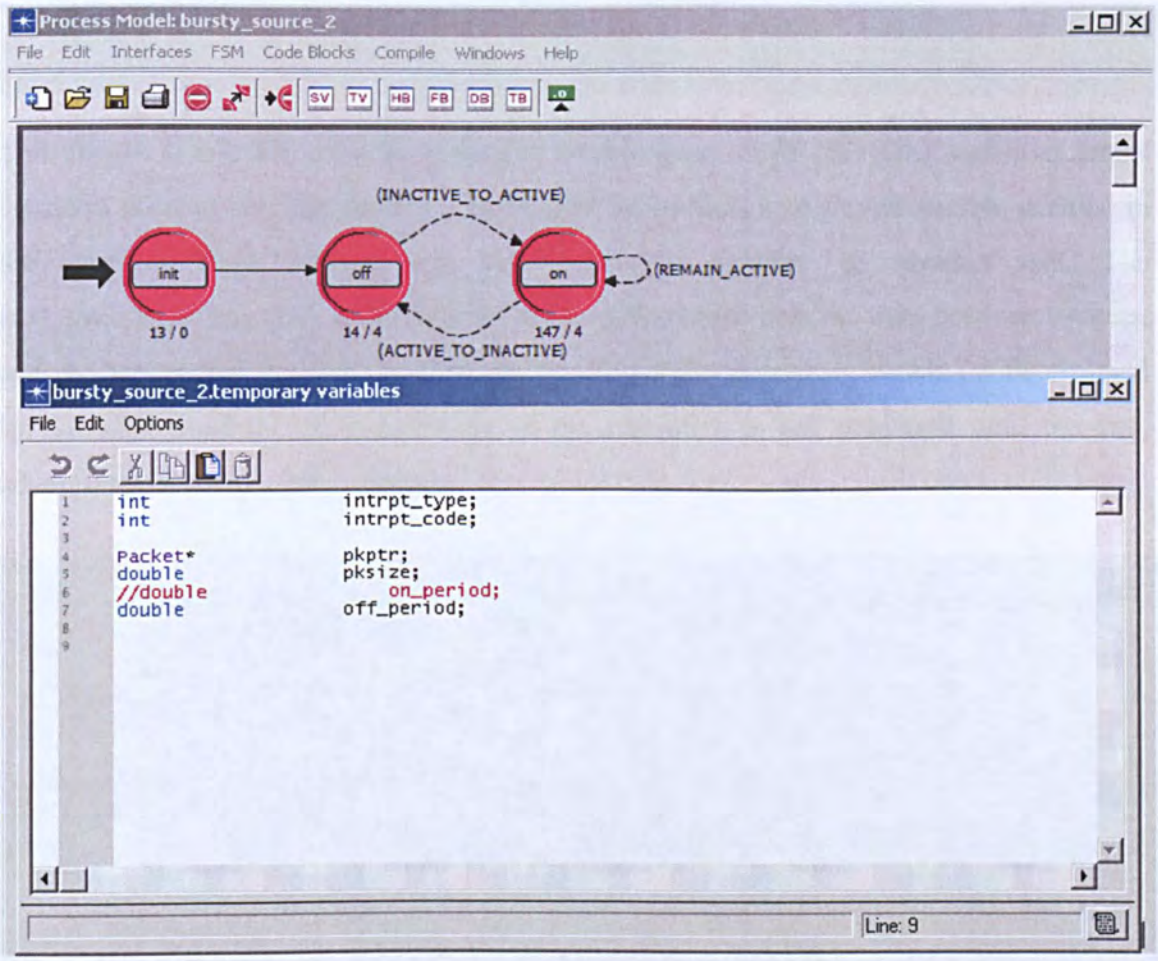


Figure 4.9 Editing temporary variables

Having created the necessary variables, the mechanism can now be imlemented by means of the state enter and exit executives. The following code appears at the beginning of the ON state's entry executive:

```
if (op_intrpt_code () == OFF_TO_ON)
{
     send_number=0;
     total_stream_send +=1;
}

next_packet_arrival_time = op_sim_time () + tbp;

if (send_number==0) pksize=p_size_1;
if (send_number==1) pksize=p_size_2;
if (send_number==2) pksize=p_size_3;
if (send_number==3) pksize=p_size_4;
if (send_number==4) pksize=p_size_5;
if (send_number==5) pksize=p_size_6;
if (send_number==6) pksize=p_size_7;
if (send_number==7) pksize=p_size_8;
```

```
if (send_number==8) pksize=p_size_9;
if (send_number==9) pksize=p_size_10;
```

This checks to see if it is the beginning of a new stream (OFF_TO_ON) and if so, resets the send_number variable to zero (the number of packets sent in this stream, defined in the function block below) and increments the number of streams sent. The next_packet_arrival_time variable is set to the current time plus the time between packets, and the packet size is set to the appropriate size for the upcoming packet (as determined by the send_number). The remainder of the executive is left unaltered until the end, where the following code is added:

```
printf ("Total stream sent out of %f is %f\n", ns, total_stream_send);
if (total_stream_send > ns)
{
     op_sim_end ("end of bahar project", "***************", "", "");
}

send_number = send_number + 1;
if (send_number< nps)
{
     op_intrpt_schedule_self (next_packet_arrival_time, ON_TO_ON);
}
else
{
     op_intrpt_schedule_self (op_sim_time (), ON_TO_OFF);
}
```

Thus if the number streams sent exceeds ns (the set number of streams) the process is terminated. Otherwise, the send_number is incremented and compared with the required number of packets per stream (nps). If the stream has not yet reached its end then and ON_TO_ON transiotion is scheduled, otherwise the process is instructed to return to the OFF state.

No additional code was added to the exit executive. However, some changes were necessary to the Function Block, which contains C/C++ language functions associated with the process which can be called from states, transitions and other blocks. The code added to the function block is listed below: lines 1 and 2 declare variables used in the executive code above, while the remaining lines provide access to the parameter values stored in state variables via the dialogue shown in Figure 4.8.

```
double  send_number, total_stream_send;
double next_packet_arrival_time;

Objid bahar_comp_attr_objid, bahar_conf_objid;
op_ima_obj_attr_get (my_id, "bahar_attributes", &bahar_comp_attr_objid);
bahar_conf_objid      =      op_topo_child      (bahar_comp_attr_objid,
OPC_OBJTYPE_GENERIC, 0);
op_ima_obj_attr_get(bahar_conf_objid,"time between packets", &tbp);
op_ima_obj_attr_get(bahar_conf_objid,"time between streams", &tbs);
op_ima_obj_attr_get(bahar_conf_objid,"number of packets in a stream",
&nps);
op_ima_obj_attr_get(bahar_conf_objid,"number of streams", &ns);
op_ima_obj_attr_get(bahar_conf_objid,"packet size 1", &p_size_1);
op_ima_obj_attr_get(bahar_conf_objid,"packet size 2", &p_size_2);
op_ima_obj_attr_get(bahar_conf_objid,"packet size 3", &p_size_3);
op_ima_obj_attr_get(bahar_conf_objid,"packet size 4", &p_size_4);
op_ima_obj_attr_get(bahar_conf_objid,"packet size 5", &p_size_5);
op_ima_obj_attr_get(bahar_conf_objid,"packet size 6", &p_size_6);
op_ima_obj_attr_get(bahar_conf_objid,"packet size 7", &p_size_7);
op_ima_obj_attr_get(bahar_conf_objid,"packet size 8", &p_size_8);
op_ima_obj_attr_get(bahar_conf_objid,"packet size 9", &p_size_9);
op_ima_obj_attr_get(bahar_conf_objid,"packet size 10", &p_size_10);
```

As for the sink module, no changes were made.

4.4. Network Scenarios

The previous section describes the packet pair/sequence generation method implemented in Opnet to simulate probing. This coming section describes the different network models that this method is applied to, which includes wired, wireless and wired-cum-wireless networks. Table 4.5 shows the Opnet parameters used in these experiments.

For the purpose of this project we only used the layer 2 devices and did not include any IP layer devices. (The study of routers will be left for future work).

Attributes	**Values**
Number of packets in stream	2
Number of streams	500
Time between packets (s)	Variable
Time between streams (s)	2
Packet sizes (byte)	Variable

Table 4.5 Probing module parameters

4.4.1. Baseline Wired Scenario

This was the most basic scenario, consisting of two Ethernet workstations called Sender and Receiver connected directly with a 10BaseT Ethernet link (Figure 4.10). The sender transmits packet streams with an adjustable "input" dispersion to the receiver, which computes their delay times and hence their "output" dispersion (i.e. their time separation at the receiver).

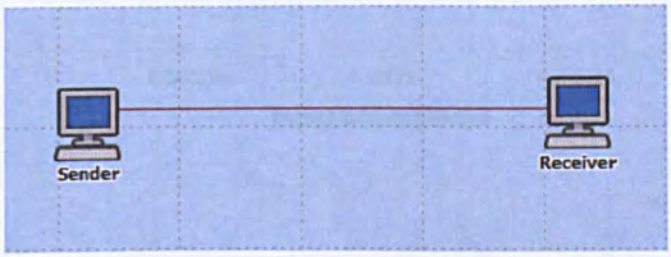
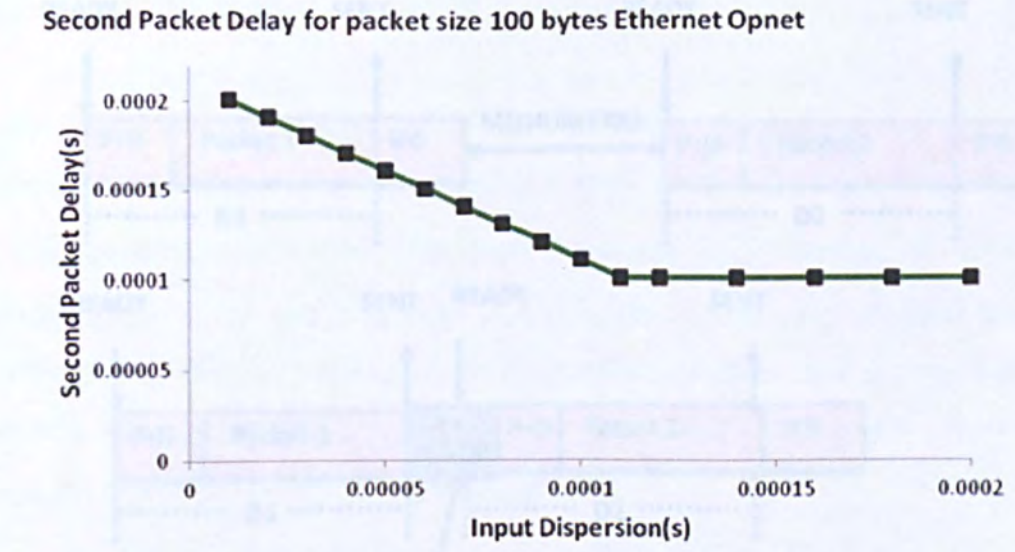


Figure 4.10 Baseline Wired Scenario

Figure 4.11 shows some typical results obtain from this model using packet pairs. The input dispersion (the time between packets at the sender) only affects the second packet delay when it is very small, and the two packets queue together in the sender's output buffer. When the input dispersion becomes larger, the medium becomes free between the two packets (Figure 4.12) and the second packet delay is constant.



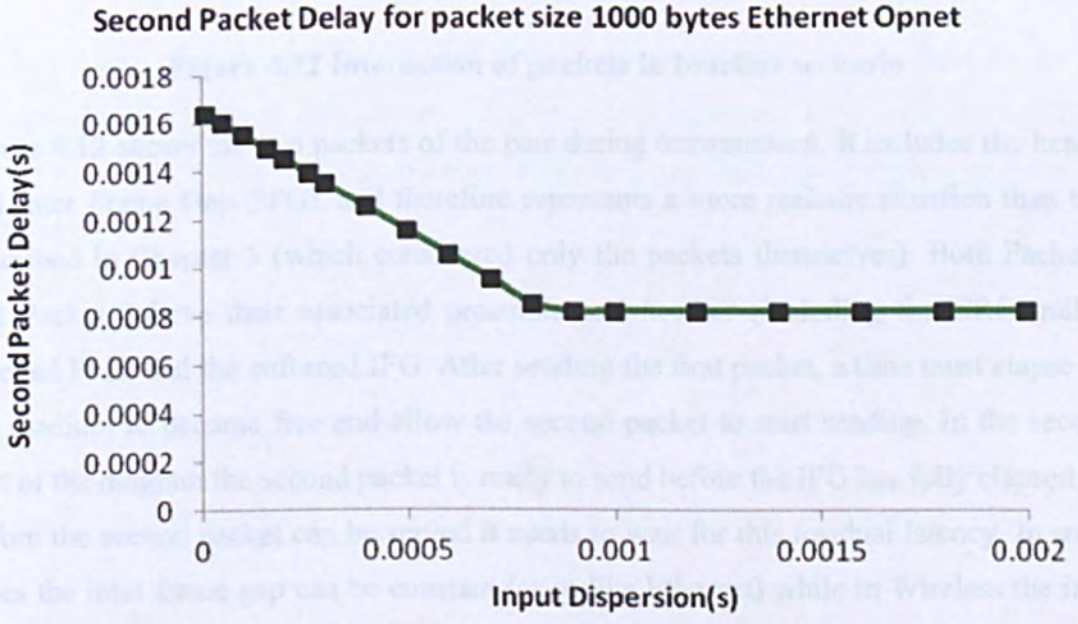


Figure 4.11 Results for based line wired scenario

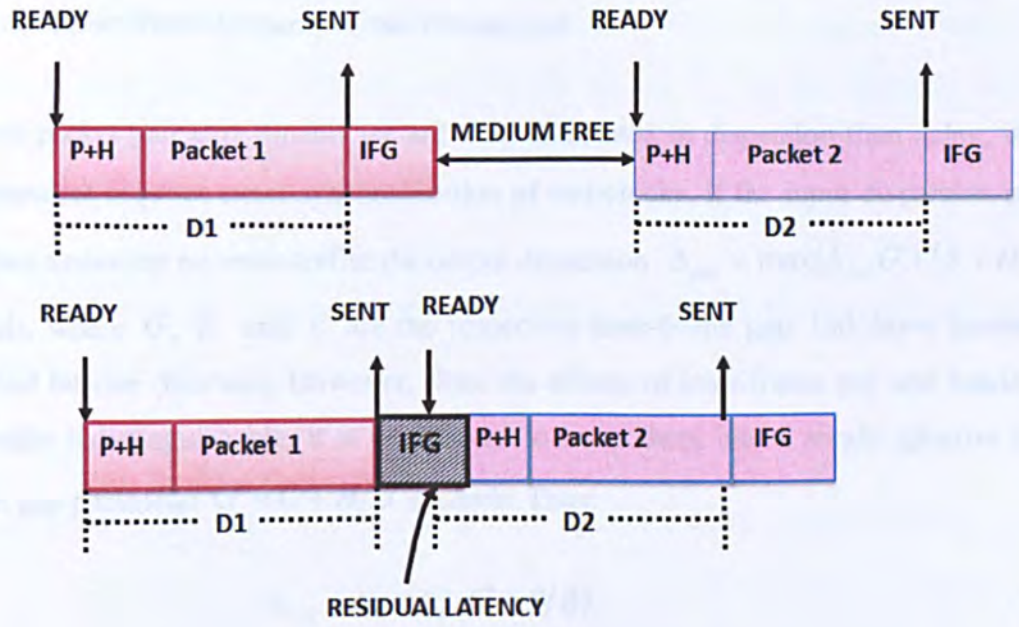


Figure 4.12 Interaction of packets in baseline scenario

Figure 4.12 shows the two packets of the pair during transmission. It includes the header and Inter Frame Gap (IFG), and therefore represents a more realistic situation than that described in Chapter 3 (which considered only the packets themselves). Both Packet 1 and Packet 2 have their associated preamble and heather (including the CRC trailer) labelled P+H, and the enforced IFG. After sending the first packet, a time must elapse for the medium to become free and allow the second packet to start sending. In the second part of the diagram the second packet is ready to send before the IFG has fully elapsed, so before the second packet can be served it needs to wait for this residual latency. In some cases the inter frame gap can be constant (as in like Ethernet) while in Wireless the inter frame gap is often variable.

The packet size for this experiment was 1000 bytes, and the Ethernet header size (including preamble) is 26 bytes, so 1026 bytes per packet are transferred. If the link rate is 10Mbit/s, then the shortest delay is 0.8208ms. This is seen as the second packet delay when the input dispersion is short. However, the IFG enforces a further 12 byte slots delay, i.e. 0.009ms, which means that the closest separation of received packets is 0.8304ms. When the input dispersion falls below this value, then the second packet delay increases, as is seen on the left of Figure 4.11.

4.4.2. Baseline Wired Dispersion and Throughput

In most packet pair experiments we are more interested in dispersion than delay, whose measurement requires exact synchronization of end-clocks. If the input dispersion is Δ_{in} sec. then assuming no cross-traffic the output dispersion $\Delta_{out} = \max(\Delta_{in}, G + (S+H)/B)$ seconds, where G, H and B are the respective inter-frame gap, link-layer header (in bits) and bit-rate (bits/sec). However, since the effects of inter-frame gap and header are externally indistinguishable, it is convenient to lump them into a single *effective* inter-packet gap parameter $G' = G + H/B$ seconds. Thus:

$$\Delta_{out} = \max(\Delta_{in}, G' + S/B). \tag{4.1}$$

As in Chapter 3, we refer to the first component within the max() function as the "independence signature" and the second as the "rate signature" [Pasztor and Veitch]. If B and G' are known then the maximum effective throughput can be computed from the formula:

$$T_{\max}(s) = Bs/(BG' + s) \tag{4.2}$$

where s is the data packet size.

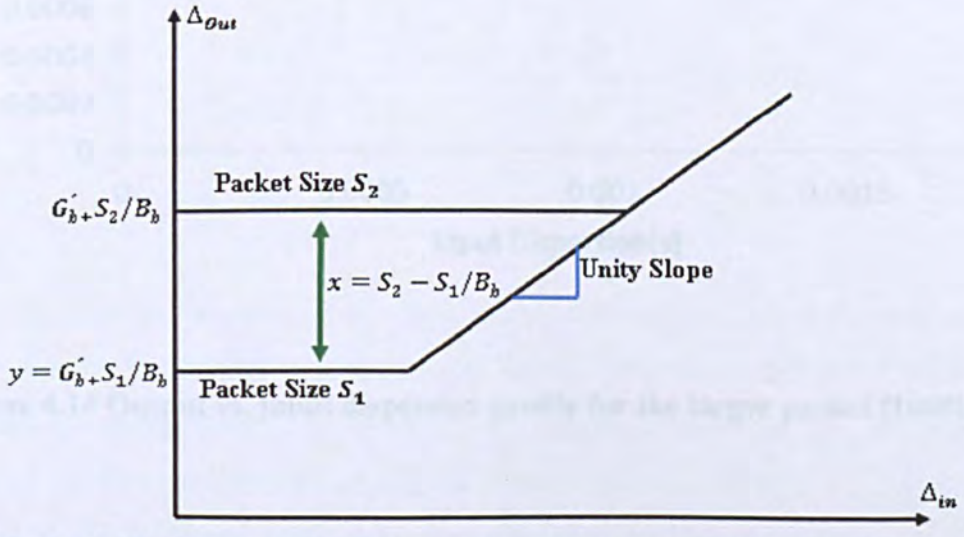


Figure 4.13 Idealized graph of output vs. input dispersion for two packet sizes. Without cross-traffic the graph is entirely dictated by the bottleneck/"narrow link".

Figure 4.13 illustrates how to obtain the parameters from the dispersion data. Two profiles are generated using different packet sizes S_1 and S_2 and the difference between the two rate signatures is denoted x. The bandwidth B can clearly be calculated:

$$B = (S_2 - S_1)/x. \tag{4.3}$$

If the rate signature for the smaller packet is denoted y, then the effective inter-frame gap can also be computed

$$G' = y - (S_1 / B). \tag{4.4}$$

On the basis of these graphs we can compute the link bandwidth using Eqn.(4.3):

$$B = \frac{S_2 - S_1}{x} = \frac{900 \times 8}{0.00072} = 10{,}000{,}000 \text{ bits/s}$$

which is the correct value for a 10BaseT link.

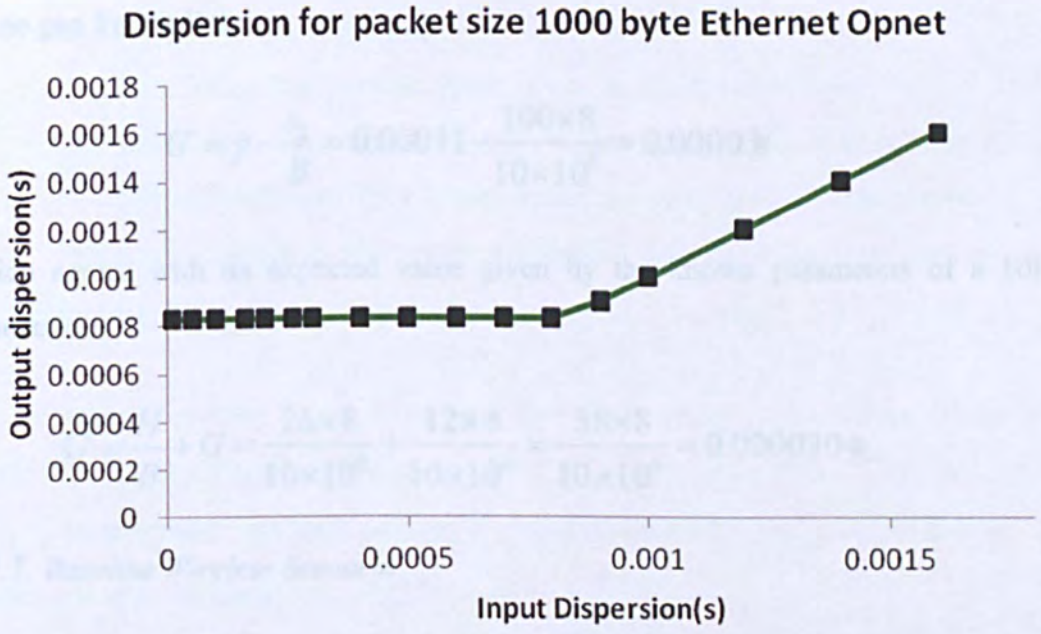


Figure 4.14 Output vs. input dispersion profile for the larger packet (1000bytes)

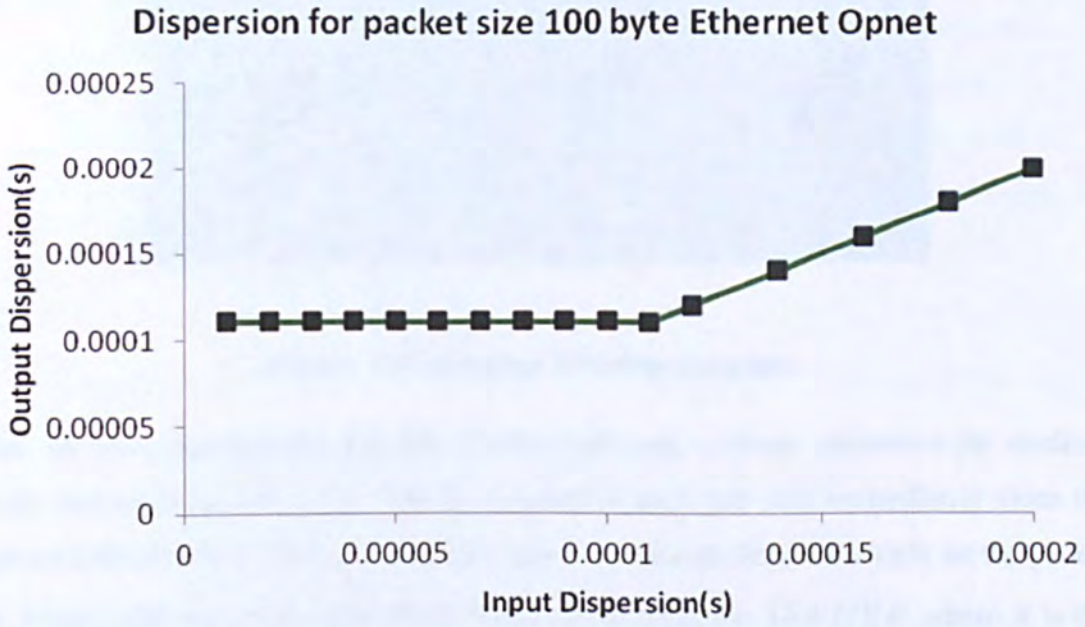


Figure 4.15 Output vs. input dispersion profile for the smaller packet (100bytes).

Using this value together with Eqn.(4.4) we can obtain a value for the effective inter-frame gap for the link:

$$G' = y - \frac{s_2}{B} = 0.00011 - \frac{100 \times 8}{10 \times 10^6} = 0.00003s$$

which agrees with its expected value given by the known parameters of a 10BaseT connection:

$$G' = \frac{H}{B} + G = \frac{26 \times 8}{10 \times 10^6} + \frac{12 \times 8}{10 \times 10^6} = \frac{38 \times 8}{10 \times 10^6} = 0.0000304s.$$

4.4.3. Baseline Wireless Scenario

This is identical to the previous scenario, with the exception that the two workstations communicate wirelessly. The Opnet model employed is the Advanced Wireless LAN Workstation, whose bursty_gen module was modified in a manner identical to that described in 4.3.4 to generate packet pairs and streams. As with the wired baseline scenario, as simple Sender and Receiver were implemented (Figure 4.16).

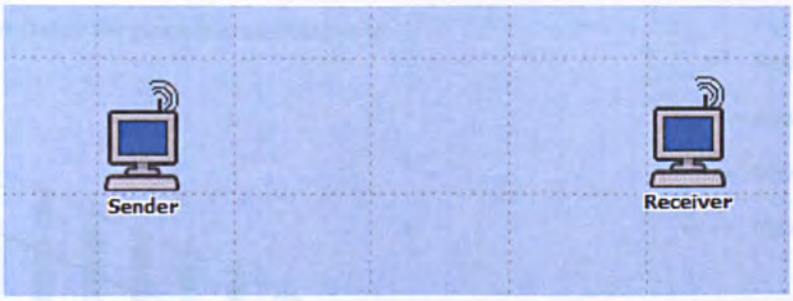


Figure 4.16 Baseline Wireless Scenario

When we have the simplest possible wireless network, without contention for medium access, and no noise corruption. The first packet in each pair sent immediately since the medium is always free. The time taken for this transmission depends simply on the packet size S (bits) and the header size H=297bytes (2376 bits), i.e. $(S+H)/B$ where B is the link bandwidth 11 Mbits/s. (Values are taken from the standard Opnet model for 802.11, see Figure 4.19.) Before another packet can be sent, several things must happen:

1. The first packet must complete transmission.
2. The SIFS (Short Inter-Frame Space, equal to 10μs) must elapses.
3. The receiver must send an ACK packet back to the sender, which takes 304 μs.
4. The DIFS (Distributed Inter-Frame Space, equal to 364 μs) must elapse.
5. The sender must wait for a further randomly-chosen "contention" period between 0 and $CW_{\max}$, where the "maximum contention window" $CW_{\max}$ is 620μs.

If two stations transmit at the same time then their transmissions interfere, there is no successful frame transmission and hence no ACK packet is sent. If this happens then $CW_{\max}$ is increased and both stations try again. However, this does not happen in this simple scenario with only one sending node.

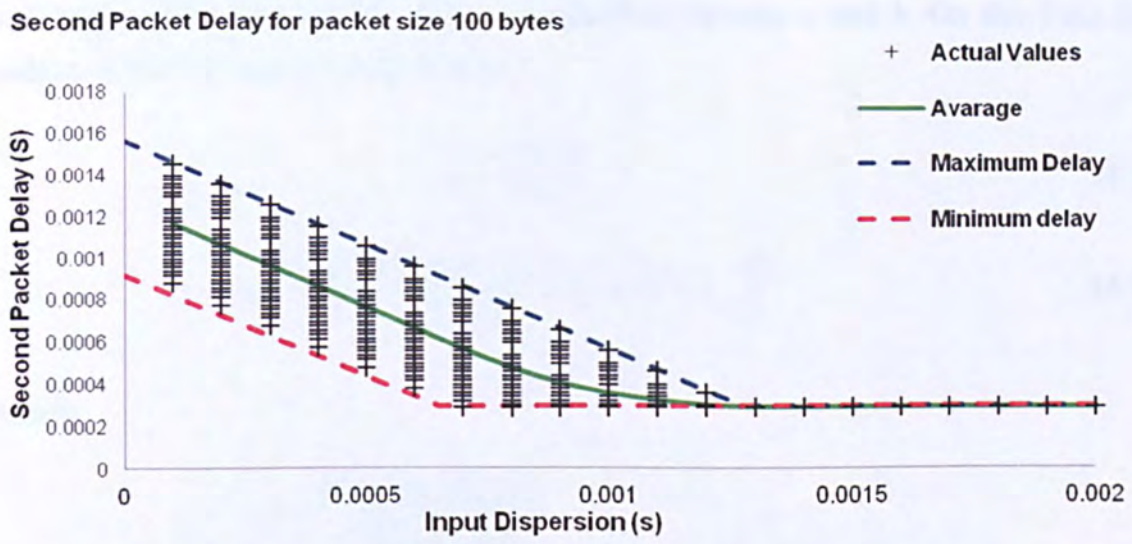


Figure 4.17 Simulated second packet delay vs. input dispersion using 100byte probe packets.

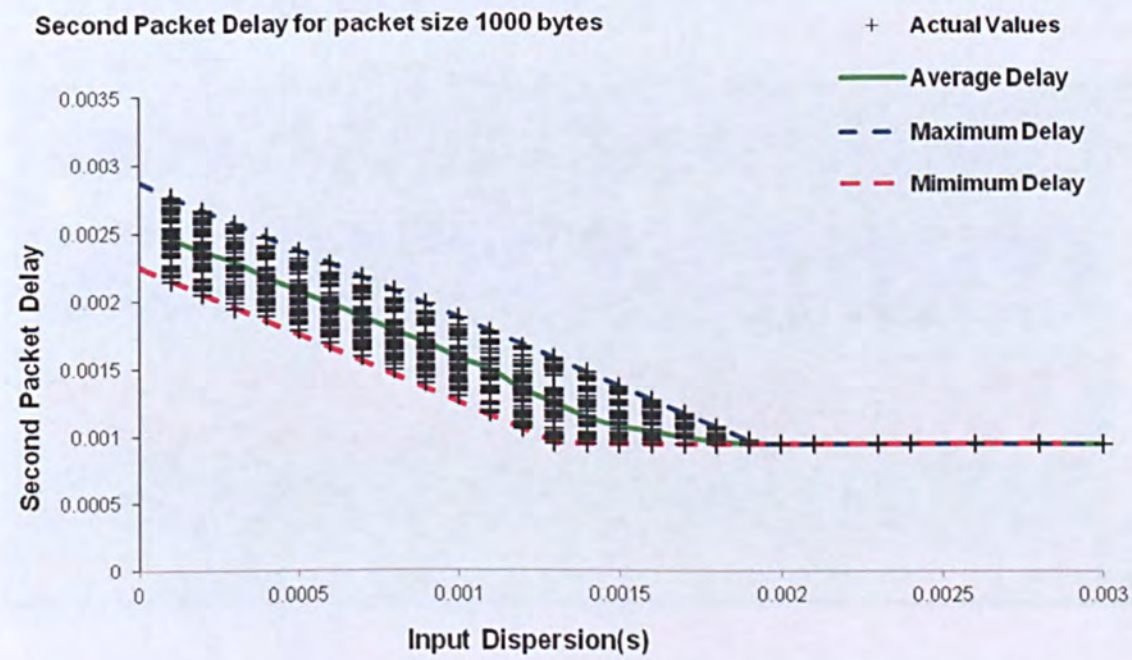


Figure 4.18 Simulated second packet delay vs. input dispersion using 1000byte probe packets.

We can represent this rather complex sequence much more simply in terms of a constant inter-frame gap $\overline{G} = SIFS + ACK + DIFS$ (where ACK is the whole time associated with ACK packet) plus a variable inter frame gap $\widetilde{G} = \text{unif}(0, CW_{\max})$ where $\text{unif}(a,b)$

indicates a value chosen with uniform probability between a and b. On this basis the delays of the two packets can be written:

$$d_1 = \frac{S+H}{B} \tag{4.5}$$

$$d_2 = \frac{S+H}{B} + \left(\frac{S+H}{B} + \overline{G} + \widetilde{G} - \Delta_{in} \right)^+ \tag{4.6}$$

where

$$H = 297 bytes$$
$$B = 11 Mbit / s$$
$$\overline{G} = SIFS + ACK + DIFS = 50 \mu s$$
$$CW_{\max} = 620 \mu s$$

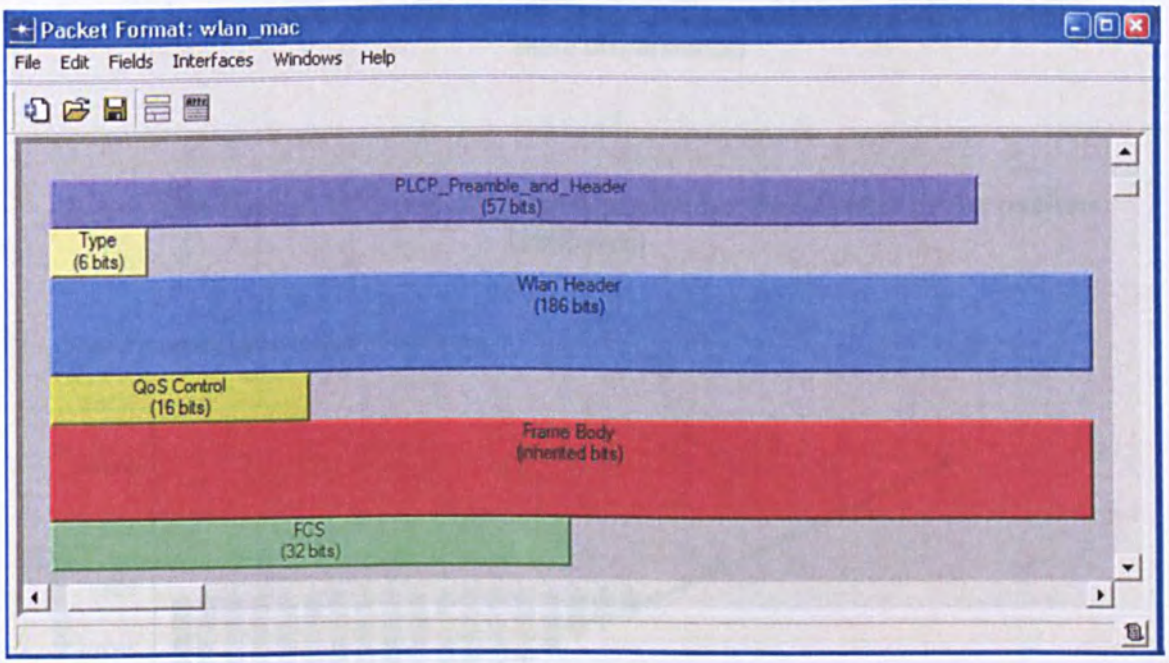


Figure 4.19 Standard 802.11 frame format in Opnet model

The maximum and minimum values predicted by Eqn.(4.7) plotted on the graphs of Figure 4.18 along with the Opnet results, showing that this model is approximately valid.

Now the output dispersion $\Delta_{out} = \Delta_{in} + (d_2 - d_1)$, and the resulting profiles are shown in Figures 4.20 and 4.21 for 100 and 1000 byte probe packets.

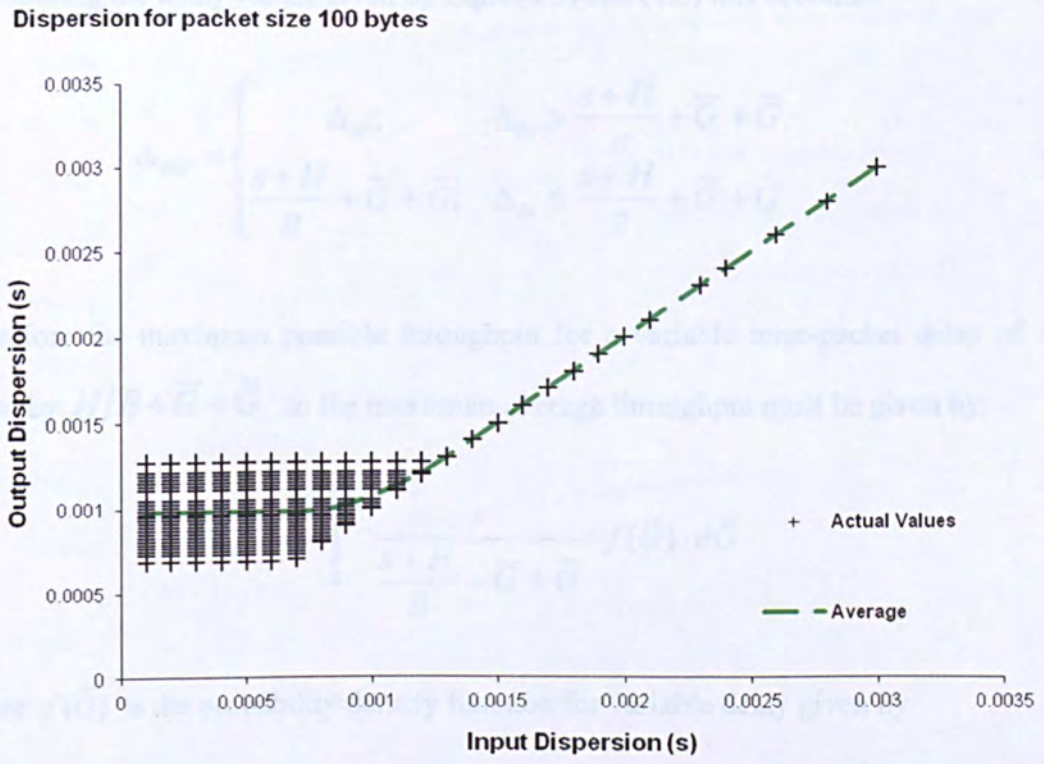


Figure 4.20 Output vs. input dispersion profile for the smaller probe packets (100bytes)

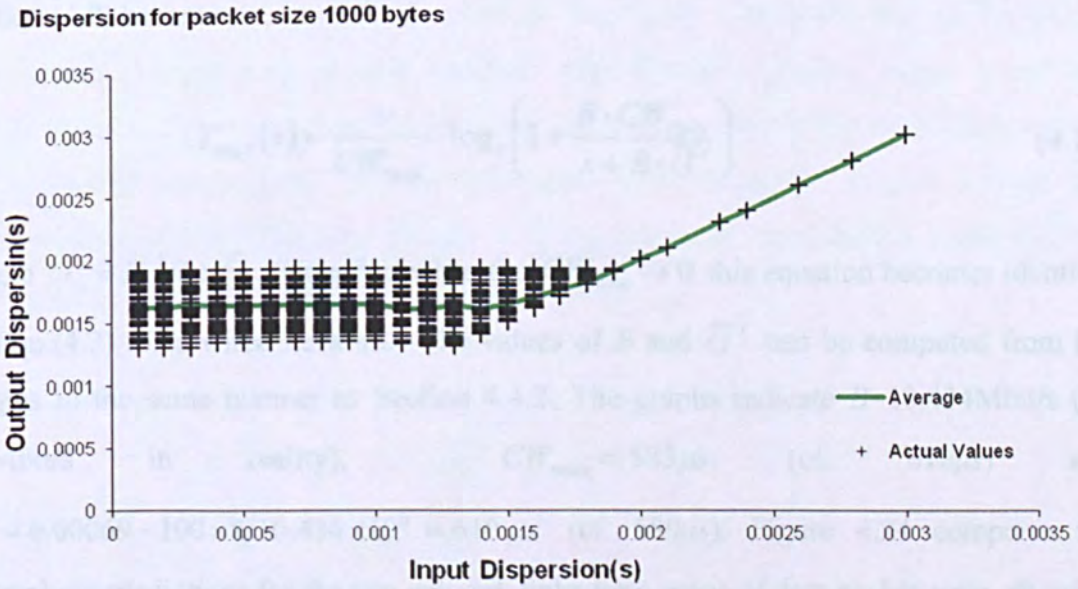


Figure 4.21 Output vs. input dispersion profile for the larger probe packets (1000bytes)

Substituting the delay values given by Eqns.(4.5) and (4.6) this becomes:

$$\Delta_{out} = \begin{cases} \Delta_{in}; & \Delta_{in} > \dfrac{s+H}{B} + \overline{G} + \widetilde{G} \\ \dfrac{s+H}{B} + \overline{G} + \widetilde{G}; & \Delta_{in} \le \dfrac{s+H}{B} + \overline{G} + \widetilde{G} \end{cases} \tag{4.7}$$

Therefore the maximum possible throughput for a variable inter-packet delay of $\widetilde{G}$ is given by $H/B + \overline{G} + \widetilde{G}$ so the maximum average throughput must be given by:

$$T_{\max} = \int_0^{CW_{\max}} \frac{s}{\dfrac{s+H}{B} + \overline{G} + \widetilde{G}} \cdot f(\widetilde{G}) \cdot d\widetilde{G} \tag{4.8}$$

where $f(\widetilde{G})$ is the probability density function for variable delay given by

$$f(\widetilde{G}) = \begin{cases} 0; \widetilde{G} < 0 \\ 1/CW_{\max}; 0 \le CW_{\max} \\ 0; X > CW_{\max} \end{cases} \tag{4.9}$$

so Eqn.(4.8) becomes:

$$T_{\max}(s) = \frac{s}{CW_{\max}} \cdot \log_e\left(1 + \frac{B \cdot CW_{\max}}{s + B \cdot \overline{G}'}\right) \tag{4.10}$$

where $\overline{G}' = H/B + \overline{G}$. (Note that when the $CW_{\max} \to 0$ this equation becomes identical to Eqn.(4.2) for a wired network.) The values of B and $\overline{G}'$ can be computed from the graphs in the same manner as Section 4.4.2: The graphs indicate B=10.434Mbit/s (cf. 11Mbit/s in reality), $CW_{\max} = 583\mu s$ (cf. $610\mu s$) and $\overline{G}' = 0.00069 - 100 \times 8/10.434 \times 10^6 = 610\,\mu s$ (cf. $580\mu s$). Figure 4.24 compares the throughput predictions for the two network links for a range of data packet sizes, showing that the predictions based on the measured parameters are practically identical to those obtained using the actual known link parameters.

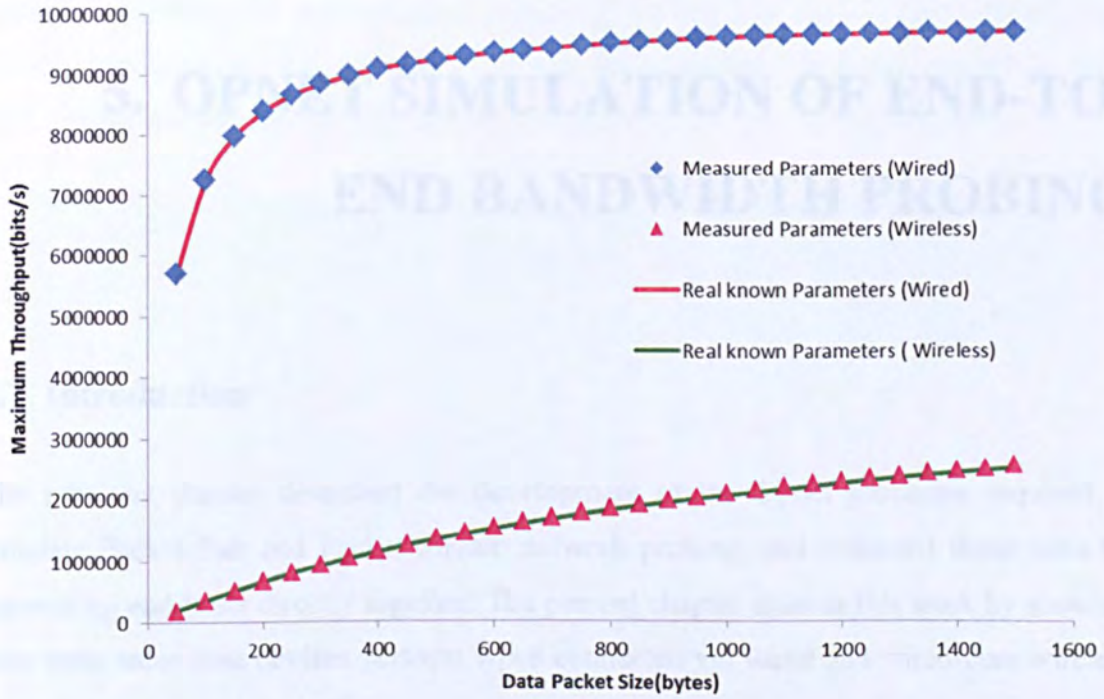


Figure 4.22 Maximum throughputs calculated from measured parameters compared with the maximum throughputs calculated from the actual parameters for wired and wireless links.

4.5. Summary and Conclusions

This chapter describes the development of the Opnet simulation testbed for packet pair/stream probing and presents baseline experiments involving single wired and wireless links. The experiments demonstrate the basic validity of the packet-pair method for bandwidth measurement, but are so far quite trivial since they assume a single link without cross-traffic. The next chapter will discuss more complicated scenarios with multiple-hops and cross traffic added to the channel.

5. OPNET SIMULATION OF END-TO-END BANDWIDTH PROBING

5.1. Introduction

The previous chapter described the development of the Opnet processes required to simulate Packet Pair and Packet Stream network probing, and validated these tools by connecting end-hosts directly together. The present chapter extends this work by showing how these same host devices perform when connected via wired and wired-cum wireless networks carrying cross-traffic.

It was decided for simplicity that these networks should exist exclusively of layer 2 devices, though the same principles would apply to a routed network. The wireless "hop" appeared either as the first stage of the probe packet's journey ("first mile") or the last stage ("last mile"), or else both the first and final stages with the wired segment in the middle. Cross traffic was generally confined to the wired network sections, though the effects of bake-offs from multiple transmitters was also examined

5.2. PPSD on a Wired Network

5.2.1. Simulated Topology

Figure 5.1 shows the basic arrangement of nodes for a wired network with cross-traffic. (This topology was chosen as it is the simplest arrangement which allows cross-traffic from another host to interact with the probe stream traffic.) The nodes labelled Sender and Receiver contain the packet pair/stream functionality, while the standard Opnet "Ethernet work station advance" model was used to implement the cross traffic sources and sinks (node 3 and node 4). The cross traffic packet size was 500 bytes, and the average inter-arrival time 0.0008s, such that the cross traffic rate was 5Mbit/s. All cross traffic arrivals were Poissonian. The two Ethernet switches in the middle were

implemented using the preconfigured Ethenrt16_switch_adv model. This network was probed using pairs of 128, 256 and 1024 byte probe packets, each pair spaced 2s apart. Each simulation was run for 20 minutes, thus covering 600 packet-pair transmissions.

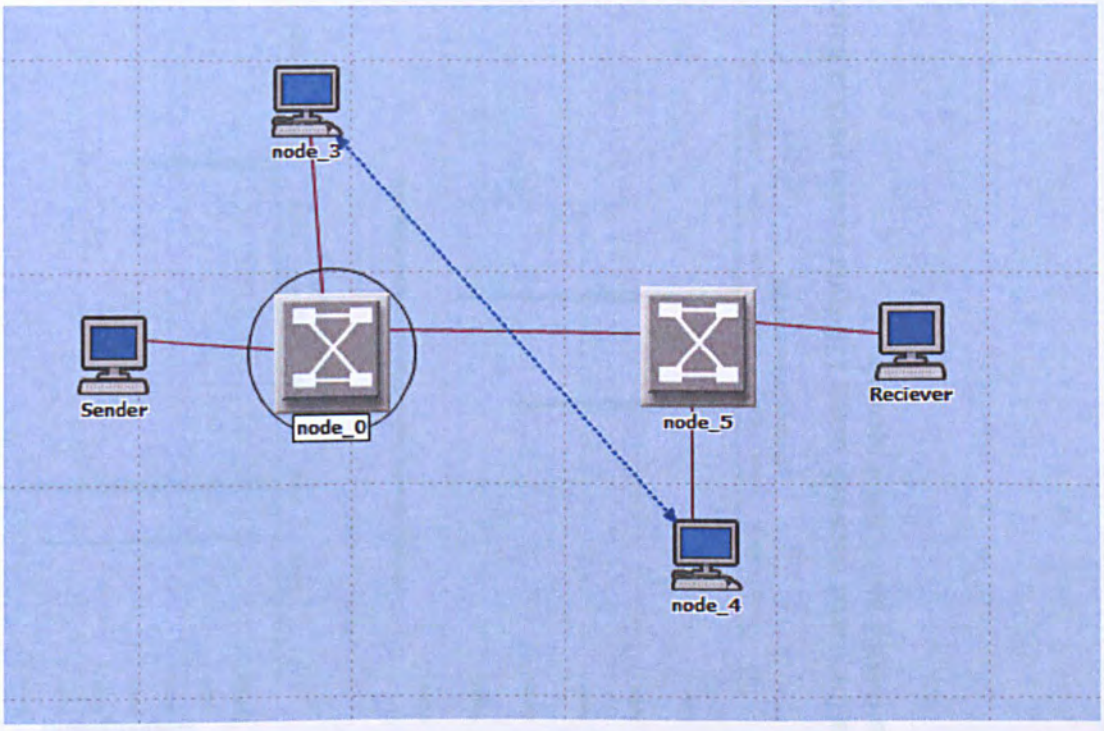


Figure 5.1 Wired network with Poisson cross-traffic, simulated in Opnet

5.2.2. Results

For each experiment the output dispersion for each packet pair was recorded, and the total results were used to estimate the probability density function using the histogram method. Figures 5.2-5.6 show the results obtained using 128-byte probe packets.

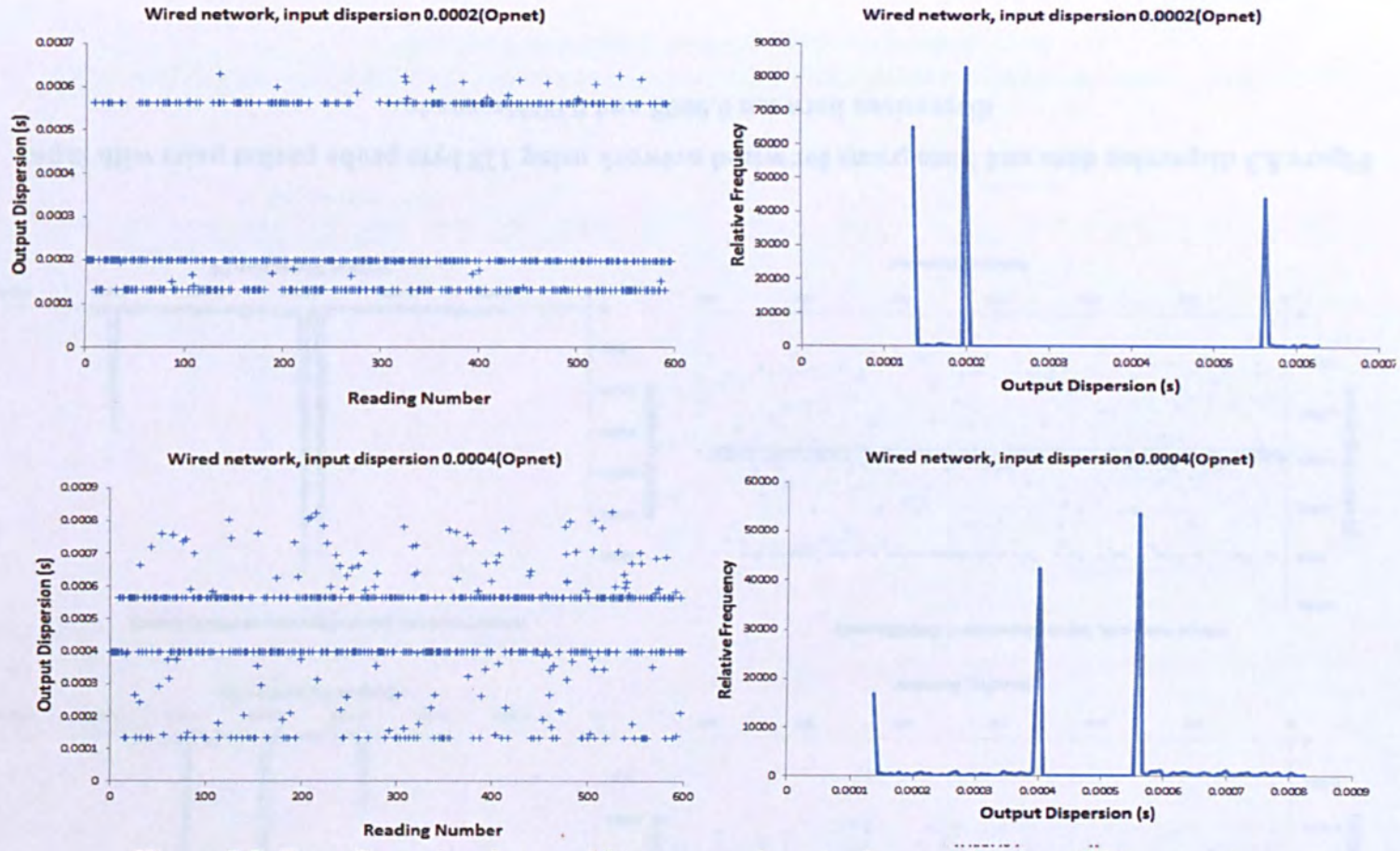


Figure 5.2 Raw dispersion data and histograms for wired network using 128-byte probe packet pairs with input dispersions 0.0002 and 0.0004 seconds.

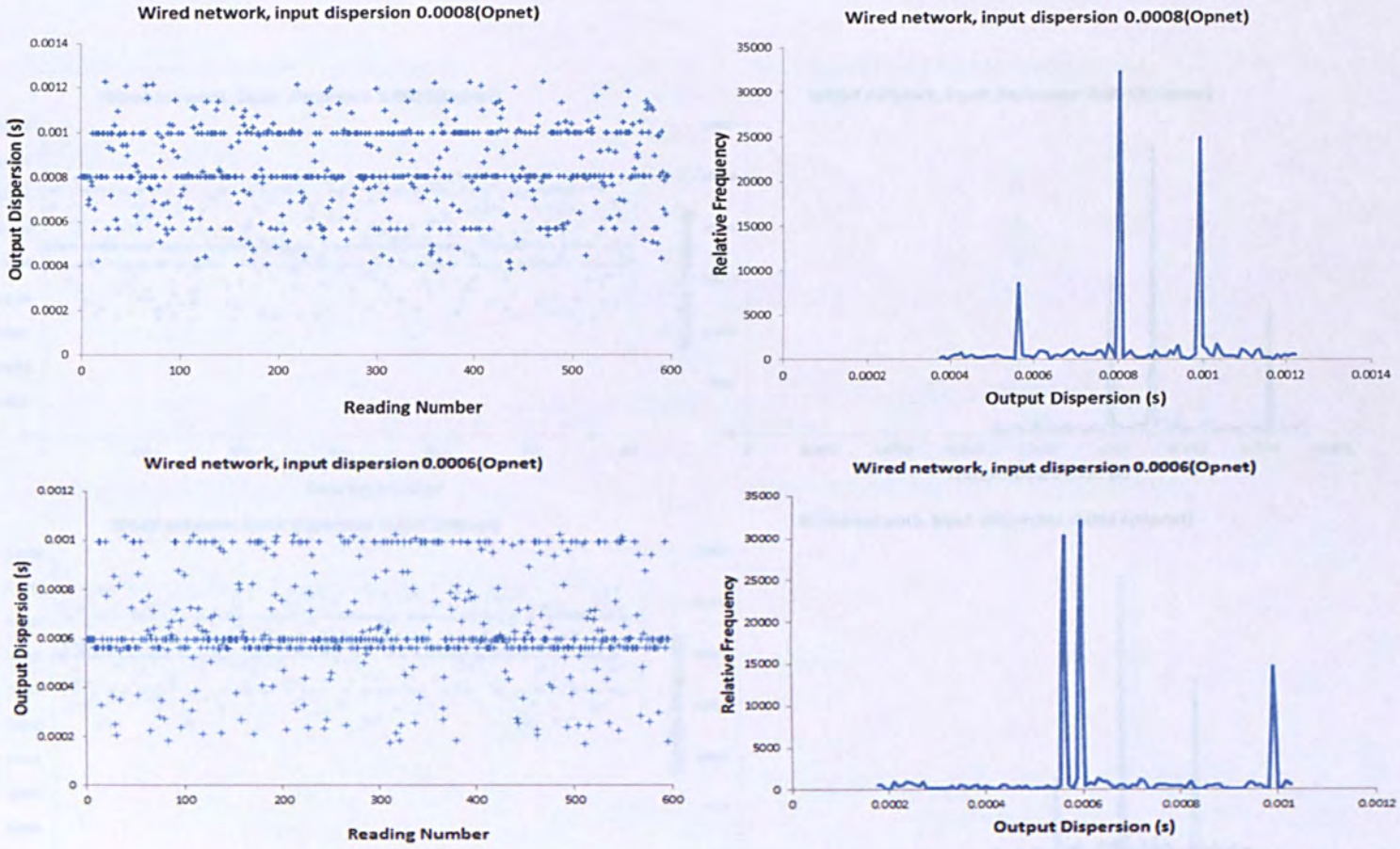


Figure 5.3 dispersion data and histograms for wired network using 128 byte probe packet pairs with input dispersions between 0.0008 and 0.0006seconds.

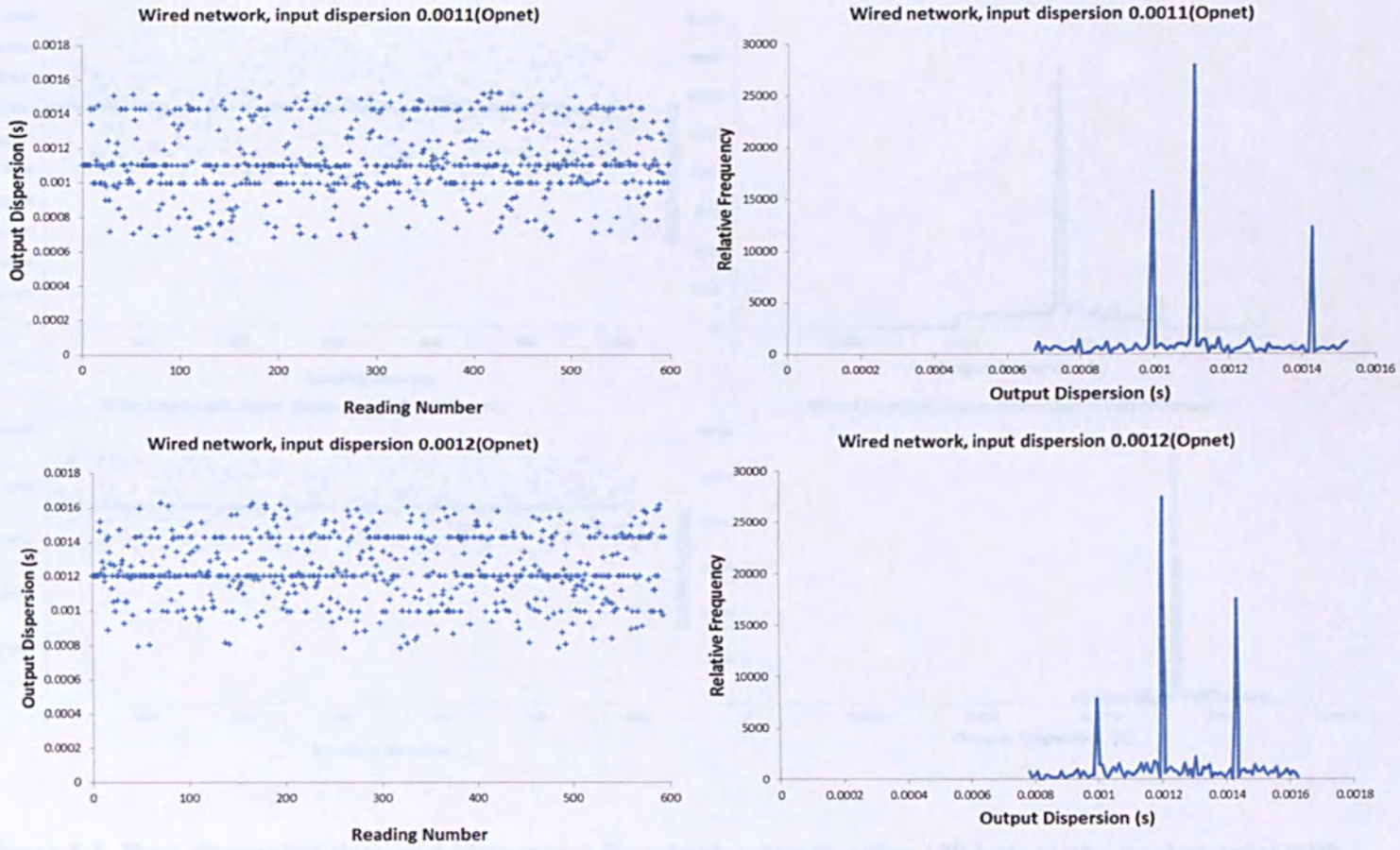


Figure 5.4 Raw dispersion data and histograms for wired network using 128 byte probe packet pairs with input dispersions between 0.0011 and 0.0012seconds

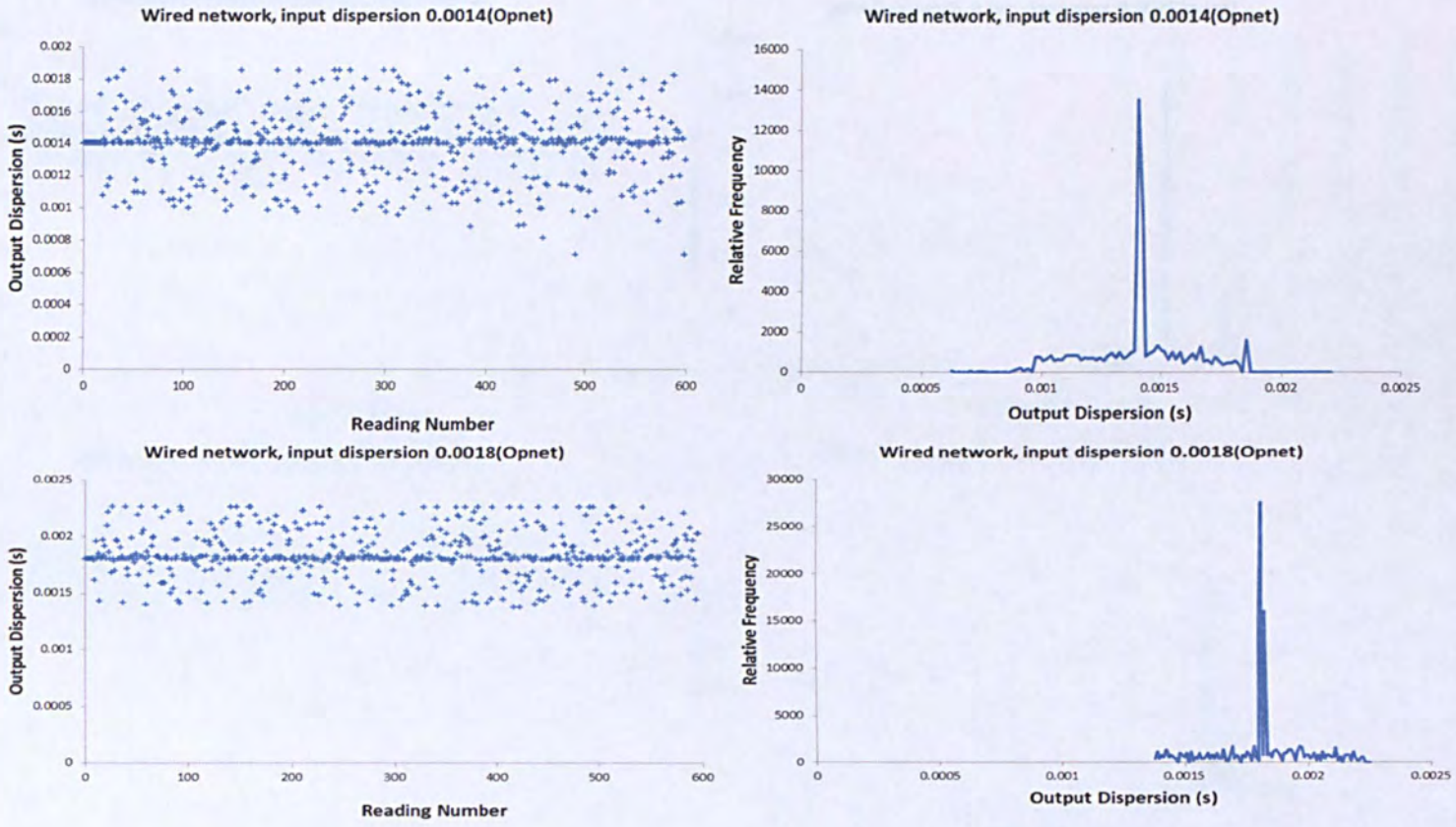


Figure 5.5 Raw dispersion data and histograms for wired network using 128 byte probe packet pairs with input dispersions between 0.0014 and 0.0018seconds.

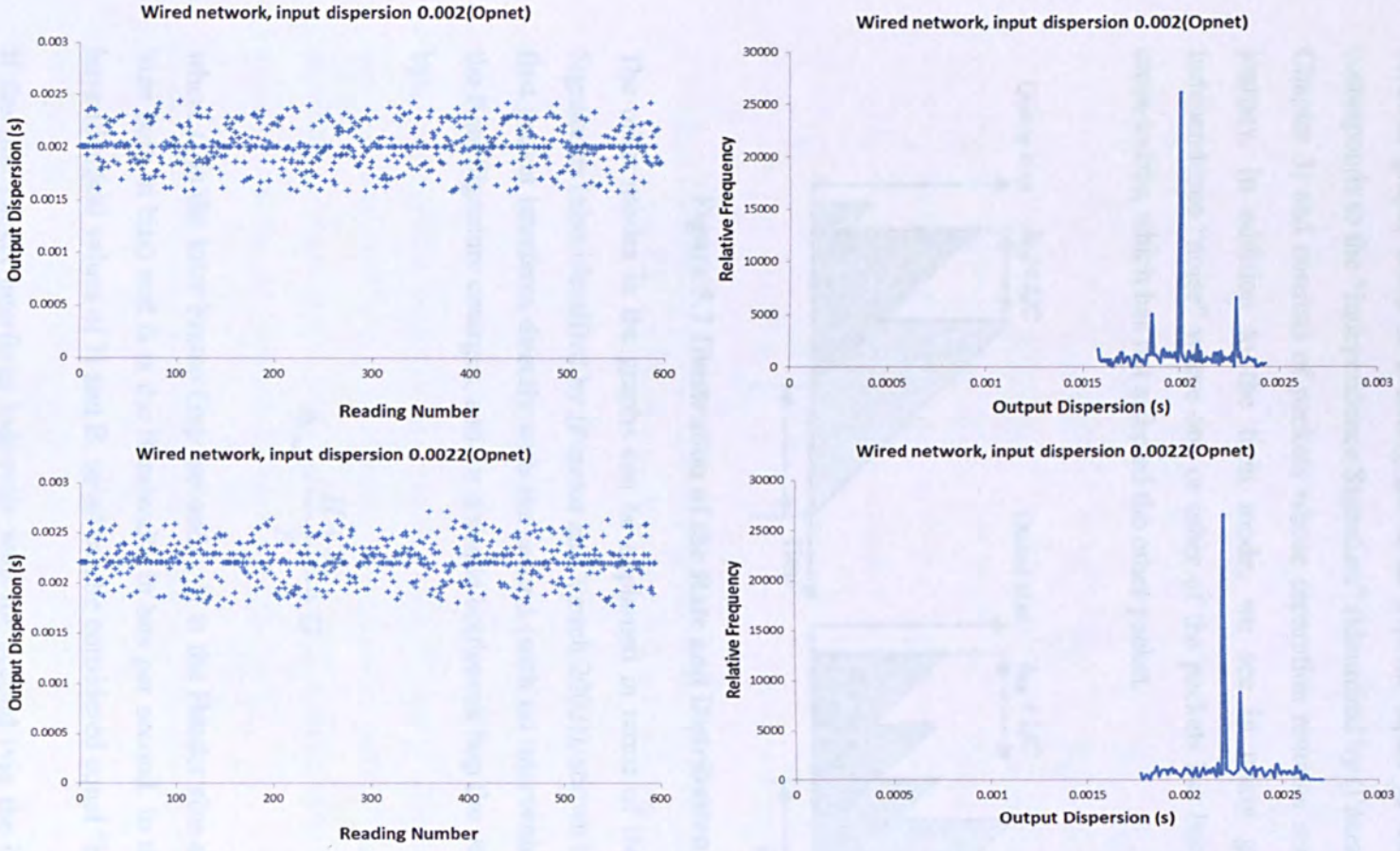


Figure 5.6 Raw dispersion data and histograms for wired network using 128 byte probe packet pairs with input dispersions between 0.002 and 0.0022seconds.

Several initial observations can be made. In all cases the data are highly modal; the data points occur in tight clusters without many data-points between them, and many of the modes change position as the input dispersion changes. The number or data-points scattered between modes increases as the input desperation increases. In all but the 0.0002s graph, there is a strong mode at a point equal to the input dispersion; this corresponds to the "Independence Signature" (identified by (Pásztor and Veitch 2002) see Chapter 3) and consists of packets whose separation remains unchanged throughout the journey. In addition to the tight mode, we see in many graphs the presence of Independence "noise" where one or other of the packets has been randomly delayed by cross-traffic, which has not affected the other packet.

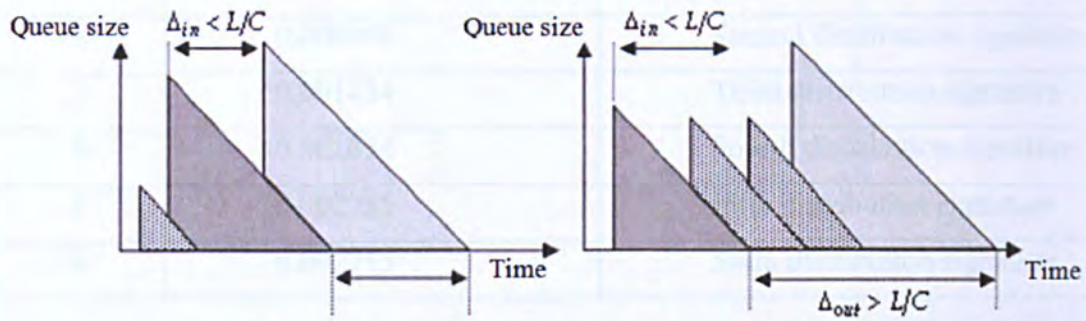


Figure 5.7 Illustration of the Rate and Distribution Signatures

The other modes in the graphs can be explained in terms of the Rate and Distribution Signatures (also identified by (Pásztor and Veitch 2002)), shown in Figure 5.7. When the first packet interferes directly with the second (with no intervening cross-traffic packets) the Rate Signature emerges, and for a single bottleneck hop the output dispersion is given by:

$$\Delta_{out} = \frac{H + S_p}{B} + G \tag{5.1}$$

where G is the Inter Frame Gap (seconds), H is the Header size and Sp the Probe Packet Size both in bits) and B is the Bandwidth in bits per second. In the present case all hops have identical values of H and B, so all can be considered equal "bottlenecks".

If the first packet interferes indirectly with the second (via the insertion of cross-traffic packets between them) this leads to the Distribution Signature. If the cross-traffic packets are of equal size then:

$$\Delta_{out} = \frac{H + S_p}{B} + G + n\left(\frac{H + S_p}{B} + G\right) \tag{5.2}$$

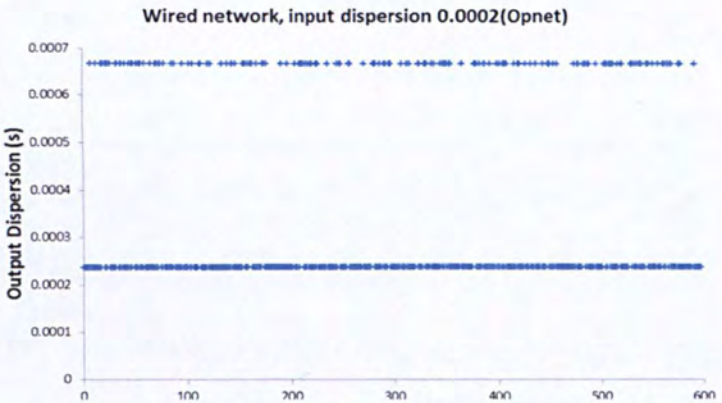
where Sc is the Cross Traffic Packet size (in bits) and n is the number of inserted cross-traffic packets. Thus n=0 represents to Rate Signature, n=1 the First Distribution Signature, n=2 the Second Distribution Signature etc.

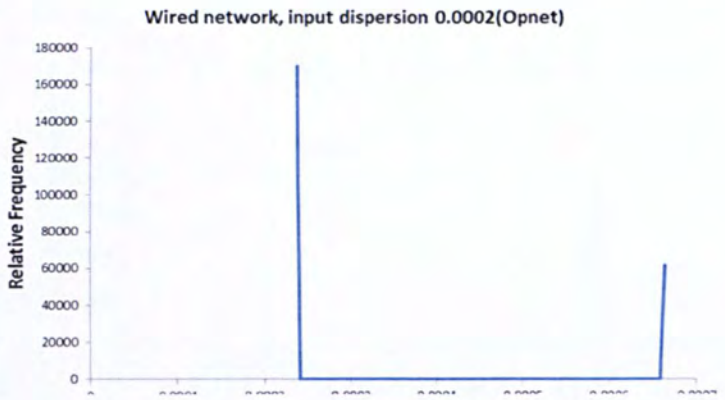
n	Output Dispersion (s)	
0	0.000133	Rate Signature
1	0.000563	First distribution signature
2	0.000994	Second distribution signature
3	0.001424	Third distribution signature
4	0.001854	Fourth distribution signature
5	0.002285	Fifth distribution signature
6	0.002715	Sixth distribution signature

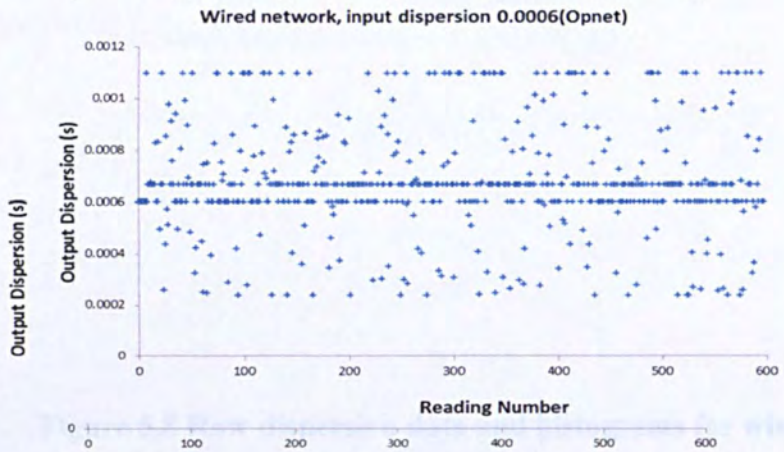
Table 5.1 Dispersion values for rate signature and first six distribution signatures for 128 byte probe packets and 500 byte cross-traffic packets

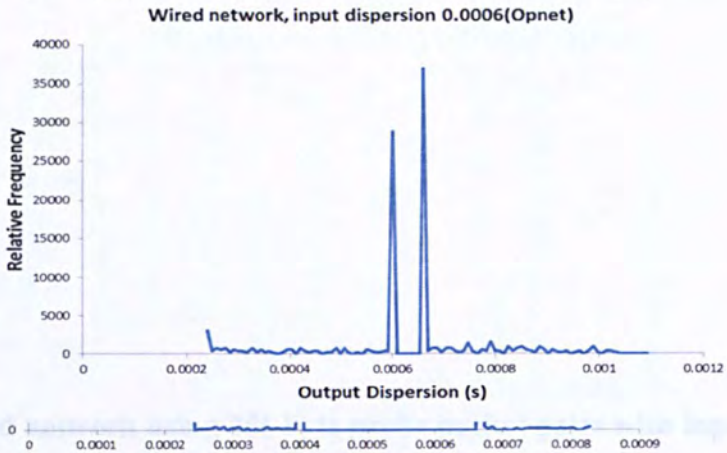
The distributions of Figure 5.2 agree well with these values, showing the independence signature (0.0002 and 0.0004s) together with the rate signature at 0.000133 and first distribution signature at 0.000563s. Figures 5.3 and 5.4 also shows the independence signature, together with the appearance of the 2nd and 3rd distribution signatures as the input gap increases. The trend continues in Figure 5.5 and 5.6; note how the lower order distribution modes vanish as the input gap increases, since more cross-traffic packets are needed to fill the gap between the probe packets.

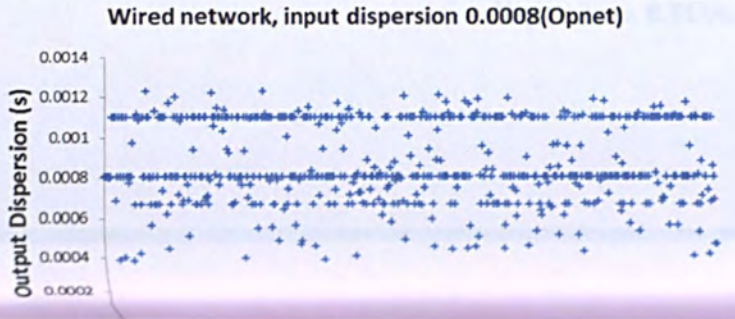
Figures 5.7-5.11 show the corresponding results obtained using 256 byte probe packets.


Wired network, input dispersion 0.0002(Opnet)
Output Dispersion (s)
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0
0
100
200
300
400
500
600


Wired network, input dispersion 0.0002(Opnet)
Relative Frequency
180000
160000
140000
120000
100000
80000
60000
40000
20000
0


Wired network, input dispersion 0.0006(Opnet)
Output Dispersion (s)
0.0012
0.001
0.0008
0.0006
0.0004
0.0002
0
0
100
200
300
400
500
600
Reading Number


Wired network, input dispersion 0.0006(Opnet)
Relative Frequency
40000
35000
30000
25000
20000
15000
10000
5000
0
0
0.0002
0.0004
0.0006
0.0008
0.001
0.0012
Output Dispersion (s)


Wired network, input dispersion 0.0008(Opnet)
Output Dispersion (s)
0.0014
0.0012
0.001
0.0008
0.0006
0.0004
0.0002

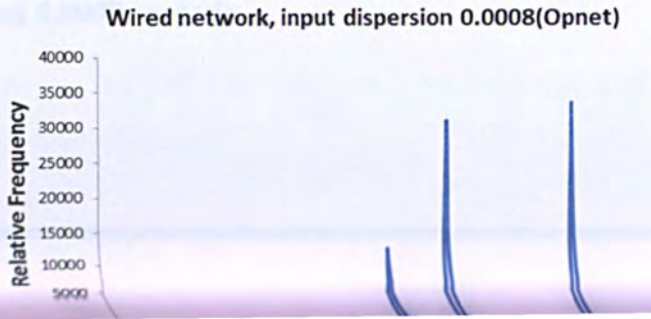

Wired network, input dispersion 0.0008(Opnet)
Relative Frequency
40000
35000
30000
25000
20000
15000
10000
5000

Figure 5.8 Raw dispersion data and histograms for wired network using 256 byte probe packet pairs with input dispersions 0.0006 and 0.0008 seconds.

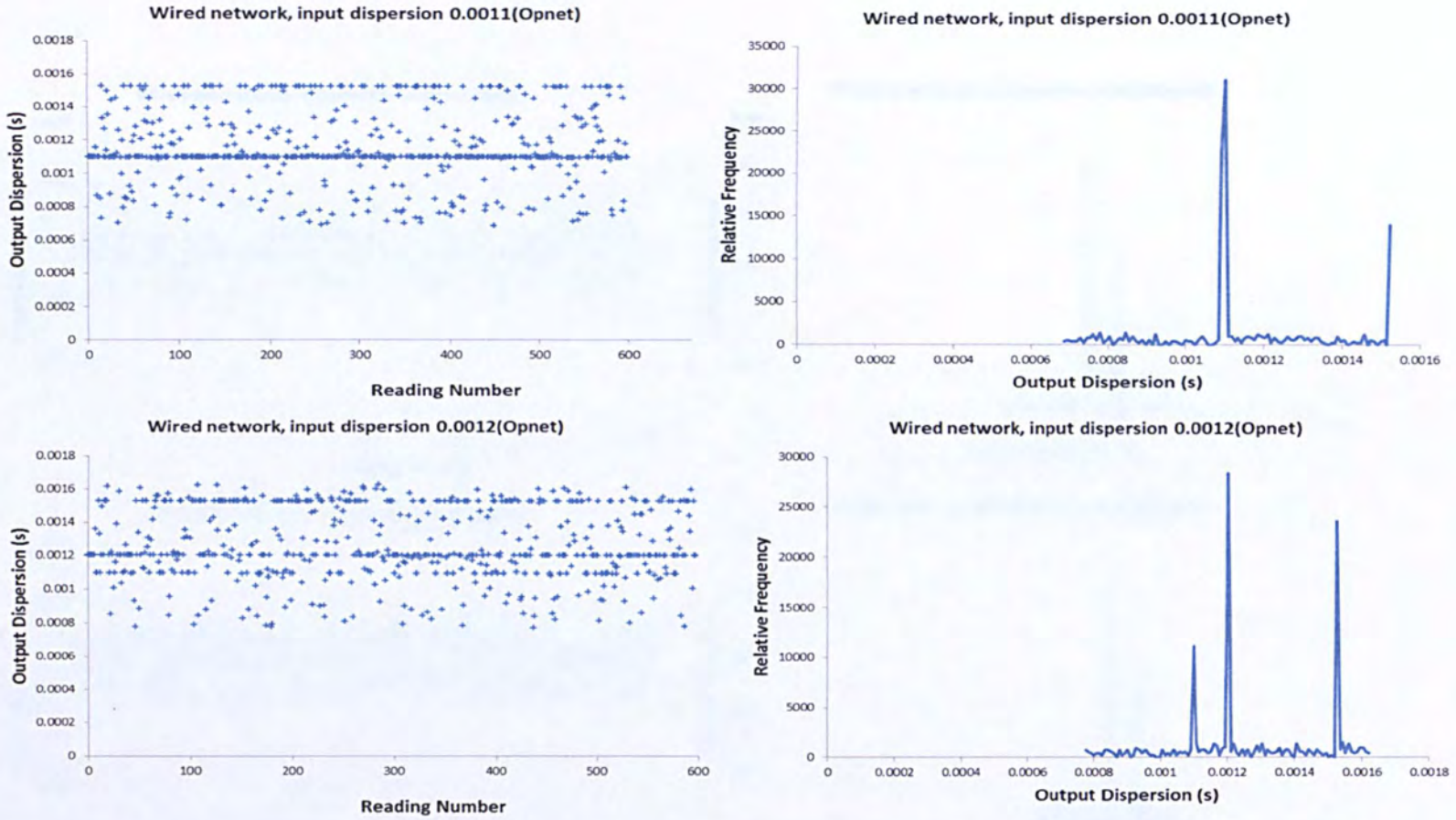


Figure 5.9 Raw dispersion data and histograms for wired network using 256 byte probe packet pairs with input dispersions 0.0011 and 0.0012seconds.

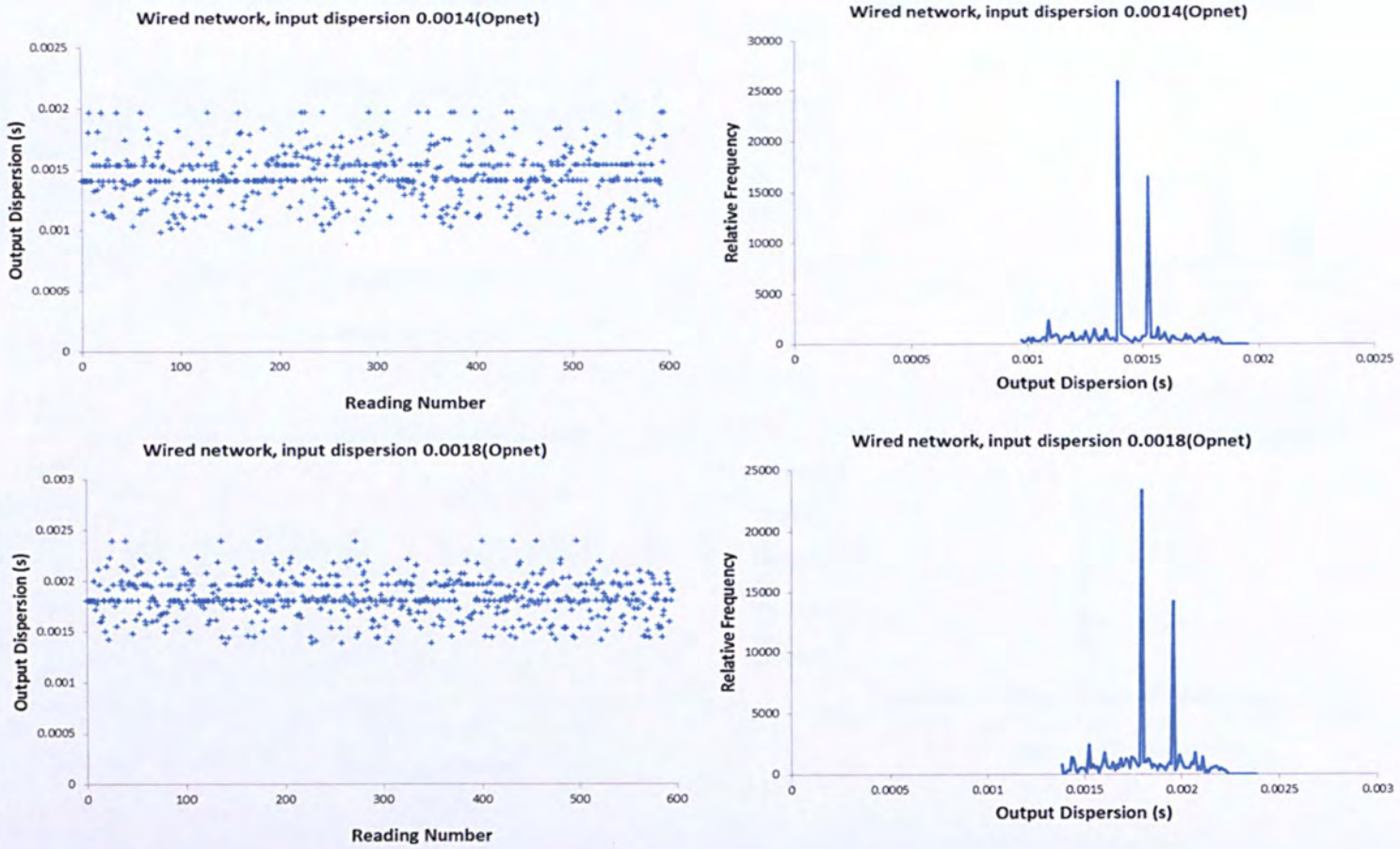


Figure 5.10 Raw dispersion data and histograms for wired network using 256 byte probe packet pairs with input dispersions 0.0014 and 0.0018seconds.

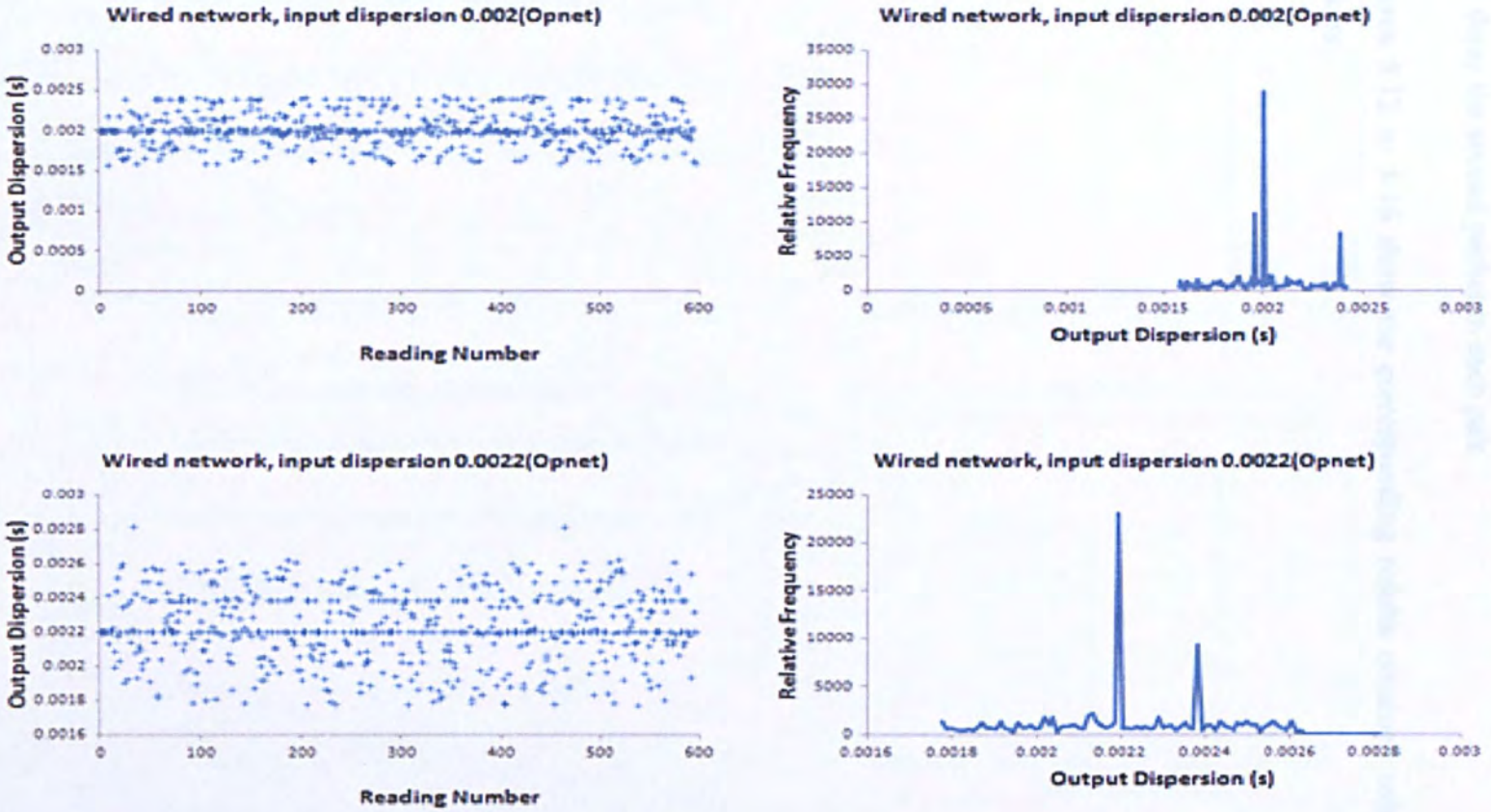


Figure5.11 Raw dispersion data and histograms for wired network using 256 byte probe packet pairs with input dispersions 0.002 and 0.0022 seconds.

The data for the 256 byte probe packets are broadly similar to the corresponding 128 byte data, but the following observation can me made: aside from the independent signature, there is a general shift of nodes to the right hand side of each histogram graph, indicating a general increase in output dispersion. This is in line with what one would expect, since the greater the packet size, the greater the likelihood that the processing of the first packet will delay the second packet in each pair.

Figures 5.12 to 5.16 show the corresponding results obtained using 1024 byte probe packets.

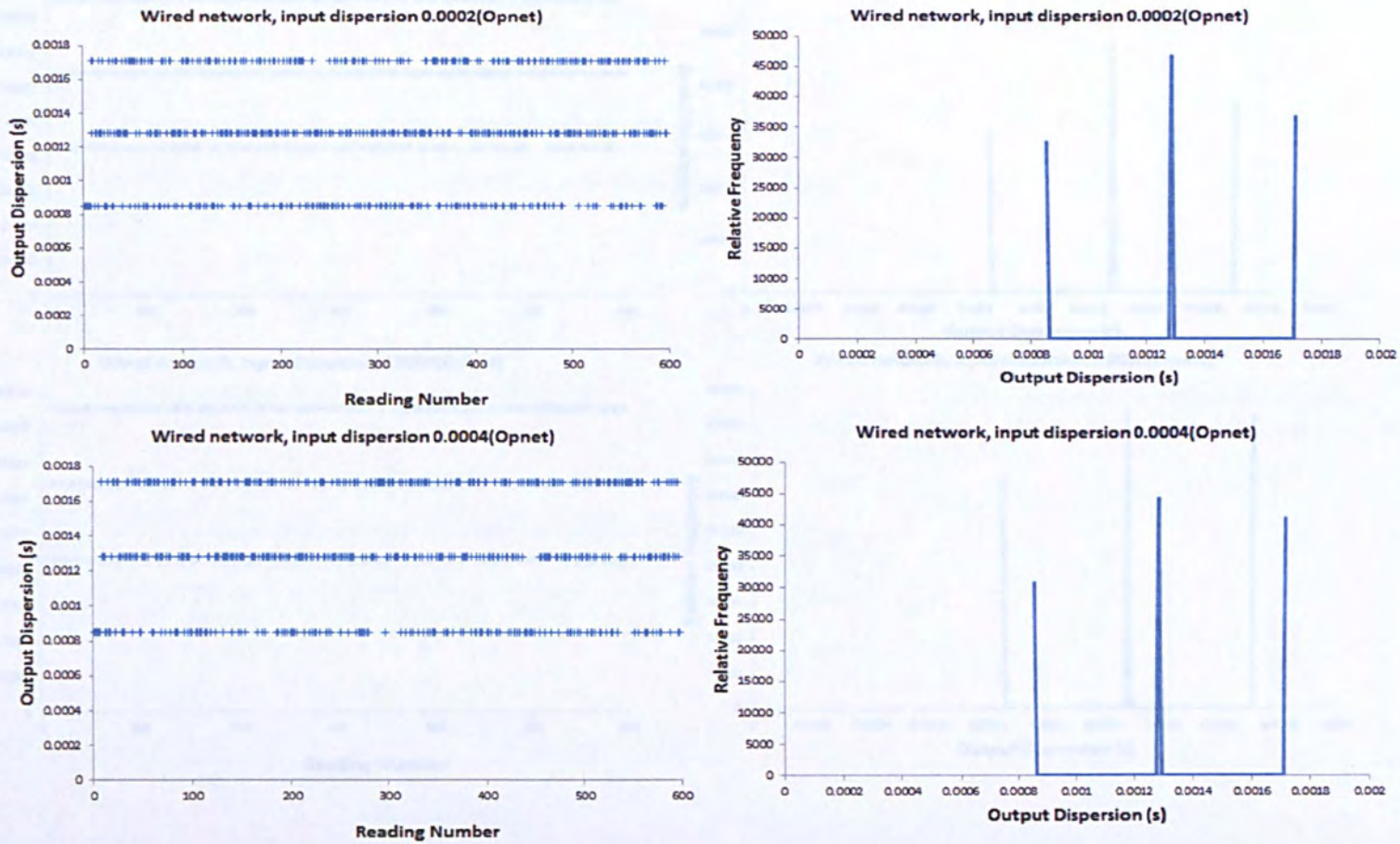


Figure 5.12 Raw dispersion data and histograms for wired network using 1024 byte probe packet pairs with input dispersions between 0.0002 and 0.0004seconds.

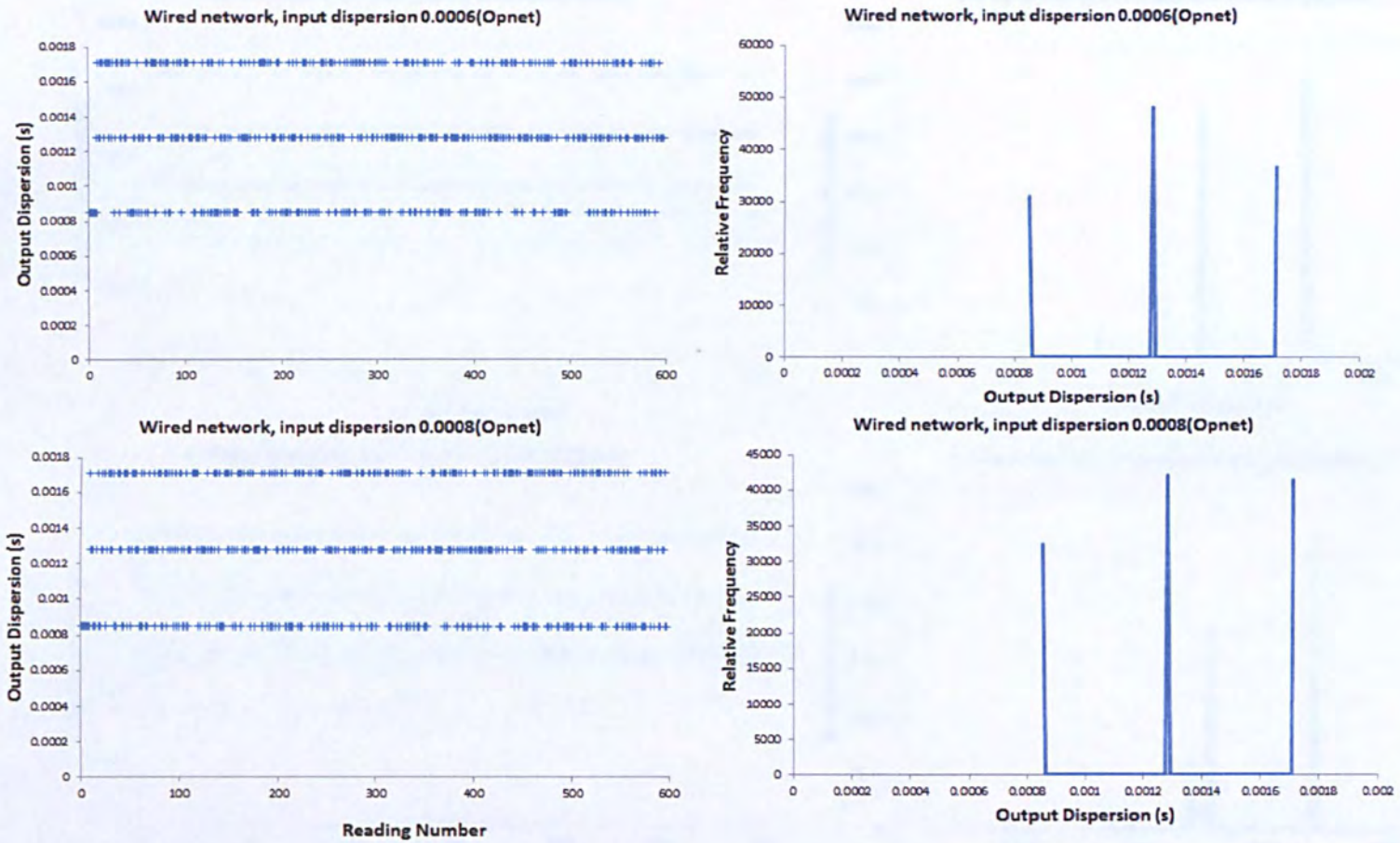


Figure 5.13 Raw dispersion data and histograms for wired network using 1024 byte probe packet pairs with input dispersions between 0.0006 and 0.0008seconds.

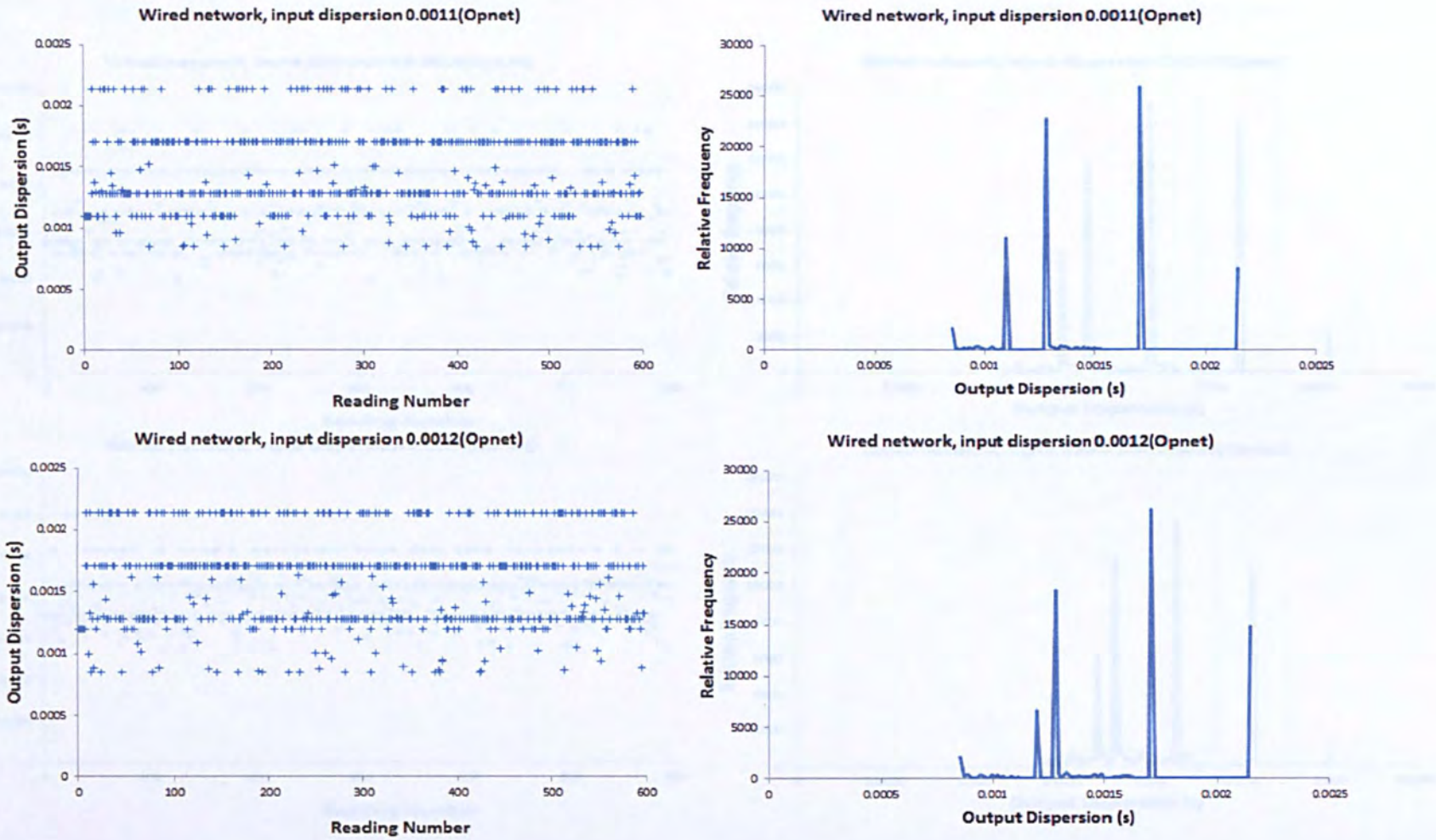


Figure 5.14 Raw dispersion data and histograms for wired network using 1024 byte probe packet pairs with input dispersions between 0.0011 and 0.0012seconds.

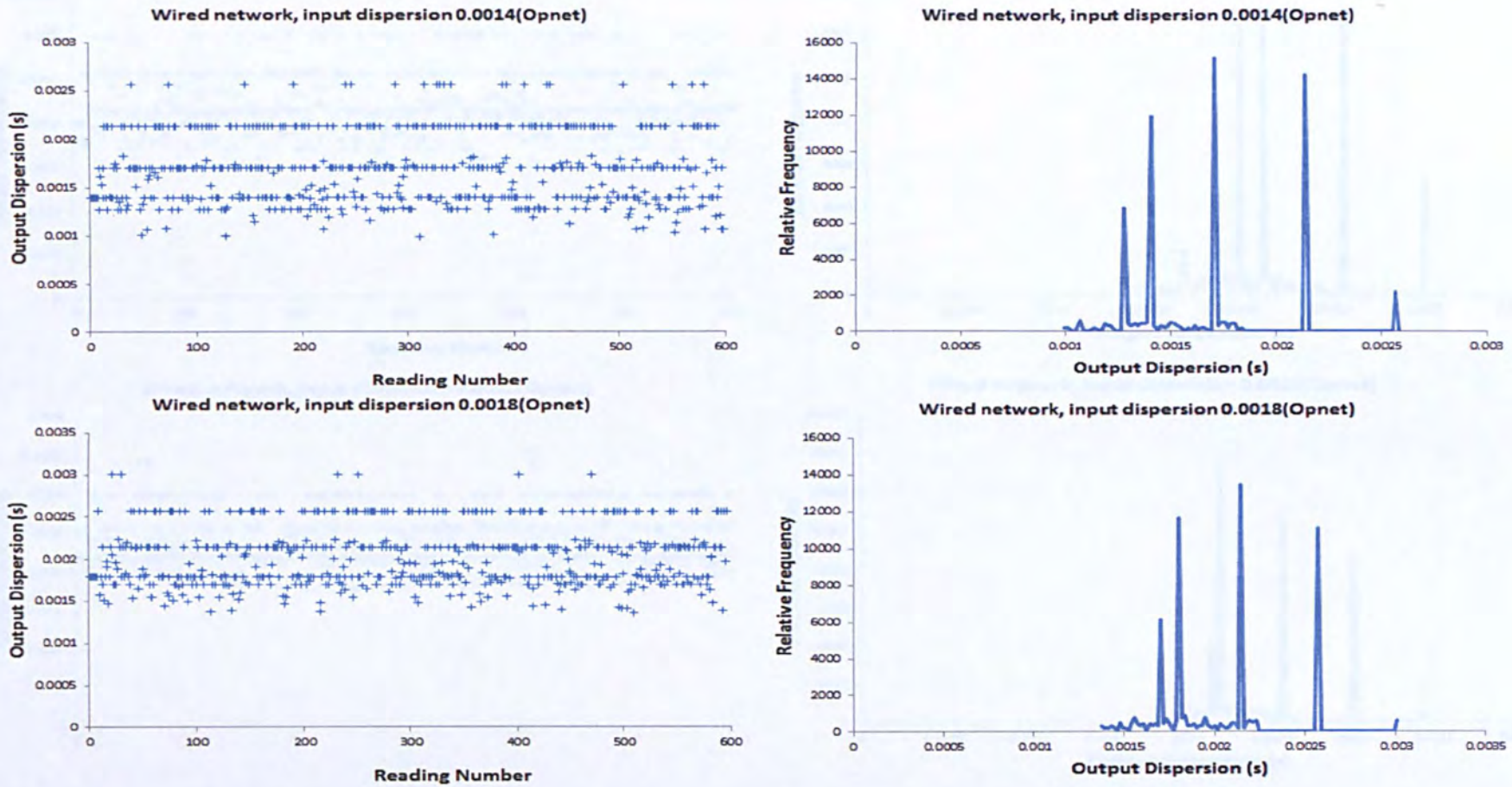


Figure 5.15 Raw dispersion data and histograms for wired network using 1024 byte probe packet pairs with input dispersions between 0.0014 and 0.0018seconds.

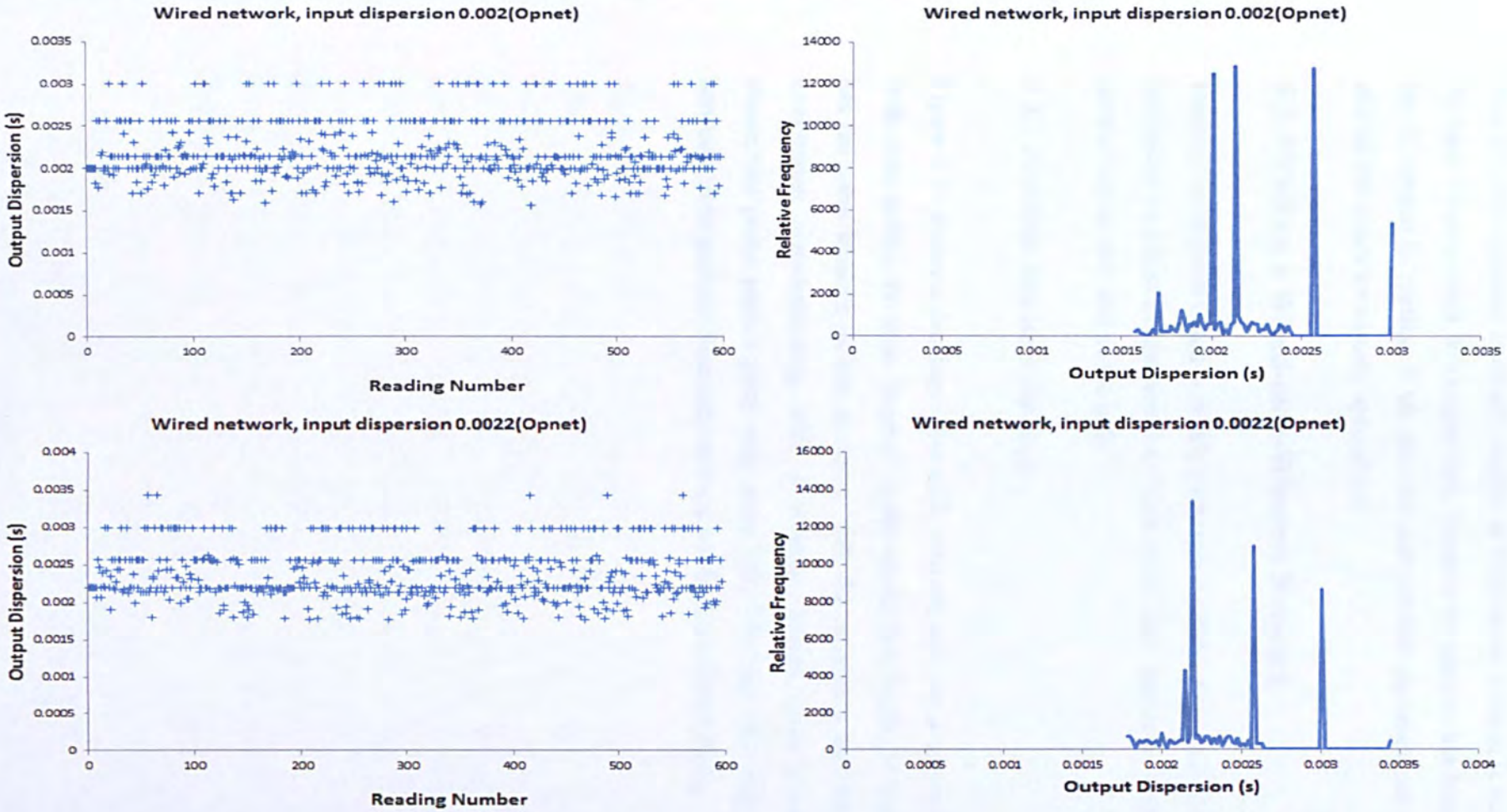


Figure 5.16 Raw dispersion data and histograms for wired network using 1024 byte probe packet pairs with input dispersions between 0.002 and 0.0022seconds.

Independent signature does not appear until the input dispersion equal 0.0011 this is how ever the rate signature is clearly visible at 0.0008496, (which is the correct value for the 10 base-T connection). We expect that, because the packets has been forced to be apart as the dispersion increasing, if we take the independent signature out of the graph it's clear that all the nodes are equally spaced out.

5.3. PPSD on a Wired-cum-Wireless Network

Having investigated wired networks, we now consider networks which include a wireless link either as a first or last hop (i.e. "first mile" and "last mile"), representing the access network at the user end of the path.

5.3.1. First Mile Simulated Topology

Figure 5.17 shows a simulated first mile network path for a wired-cum-wireless network with cross traffic, the node "source" is the sender and "node_1" is receiver. The network has the same features as last scenario, with the exception that the sender is a wireless workstation communicating with a wireless access point ("node_6"). The sender transmitted probe packet pairs with sizes 128, 256 and 1024 bytes, with a 2s interval between probe packets. Simulations were run for 20 minutes each.

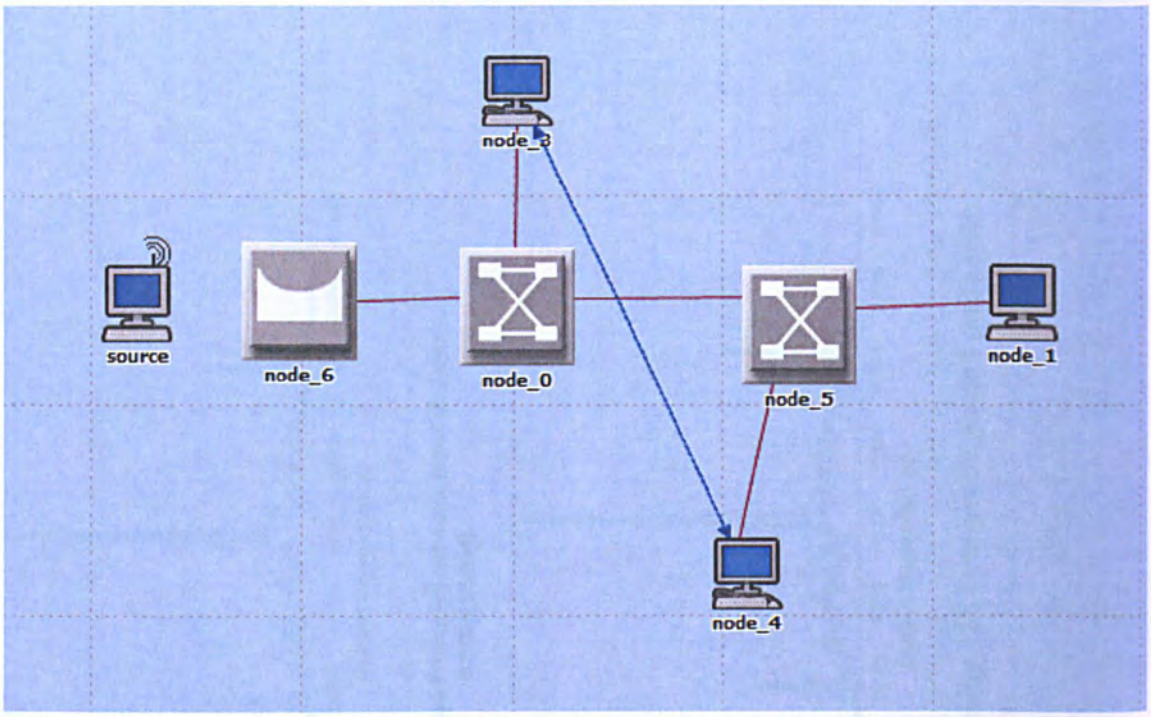


Figure 5.17 Wired-cum-Wireless network with Poisson cross-traffic, simulated in Opnet

5.3.2. First Mile Results

Figures 5.18 to 5.22 show the simulation results for the first mile wired-cum-wireless scenario. The output dispersion distribution for 0.0002s input dispersion shows a central mode at 0.001 seconds with smaller equally spaced nodes above and below it. These can be seen to represent the 1st, 2nd and 3rd distribution signature modes associated with the link between node_0 and node_5 where the probe-pairs are affected by cross-traffic (see Table 5.2). In contrast to the corresponding wired network distribution (Figure 5.2) there is a considerable number of data points between the modes which (due to the small input dispersion) cannot be attributed to independence noise. This noise must be created in the wireless link, by the variable inter-frame time associated with random back-off (see Section 4.4.3).

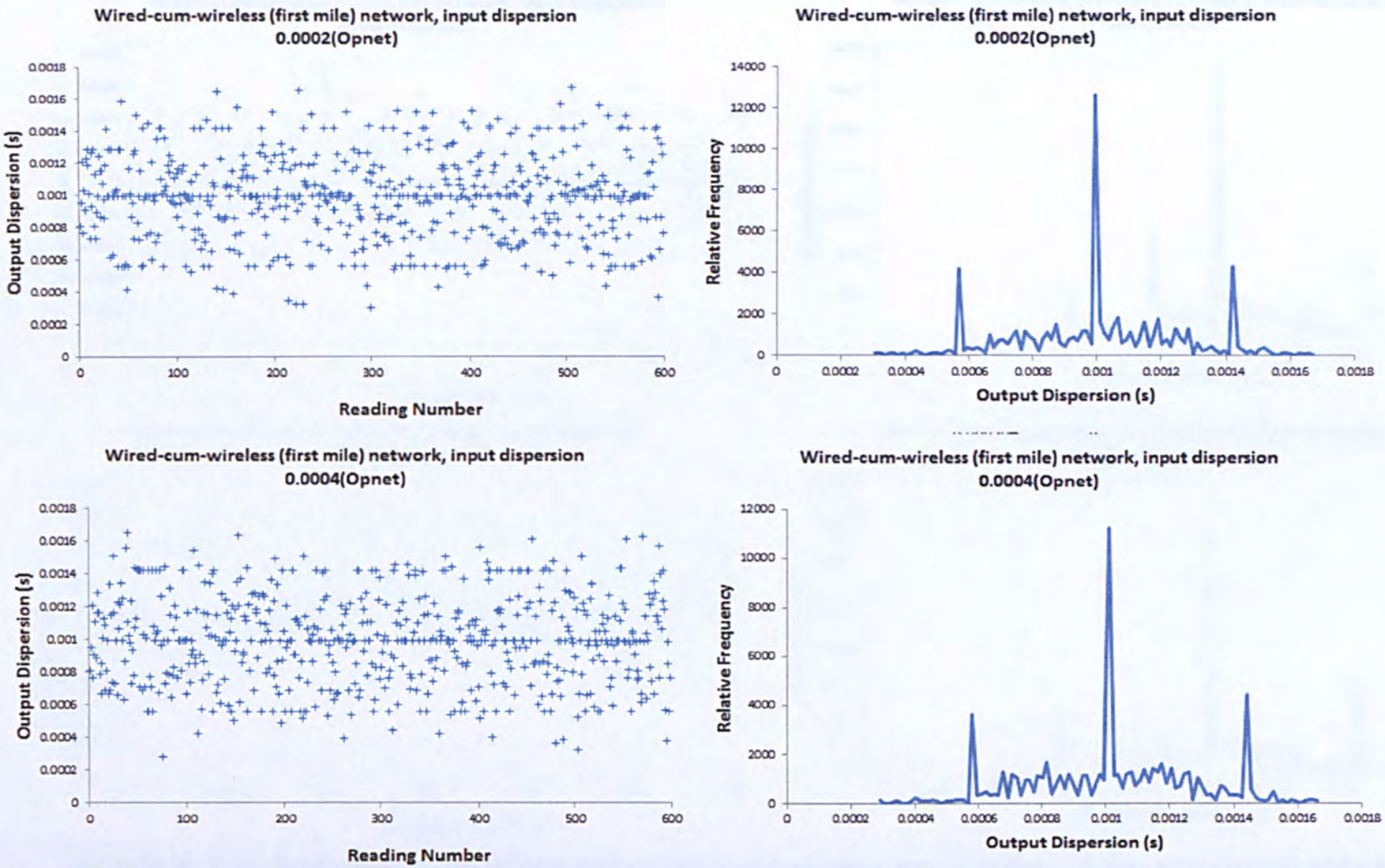


Figure 5.18 Raw dispersion data and histograms for wired-cum-wireless network using 128 byte probe packet pairs with input dispersions for 0.0002 and 0.0004 seconds.

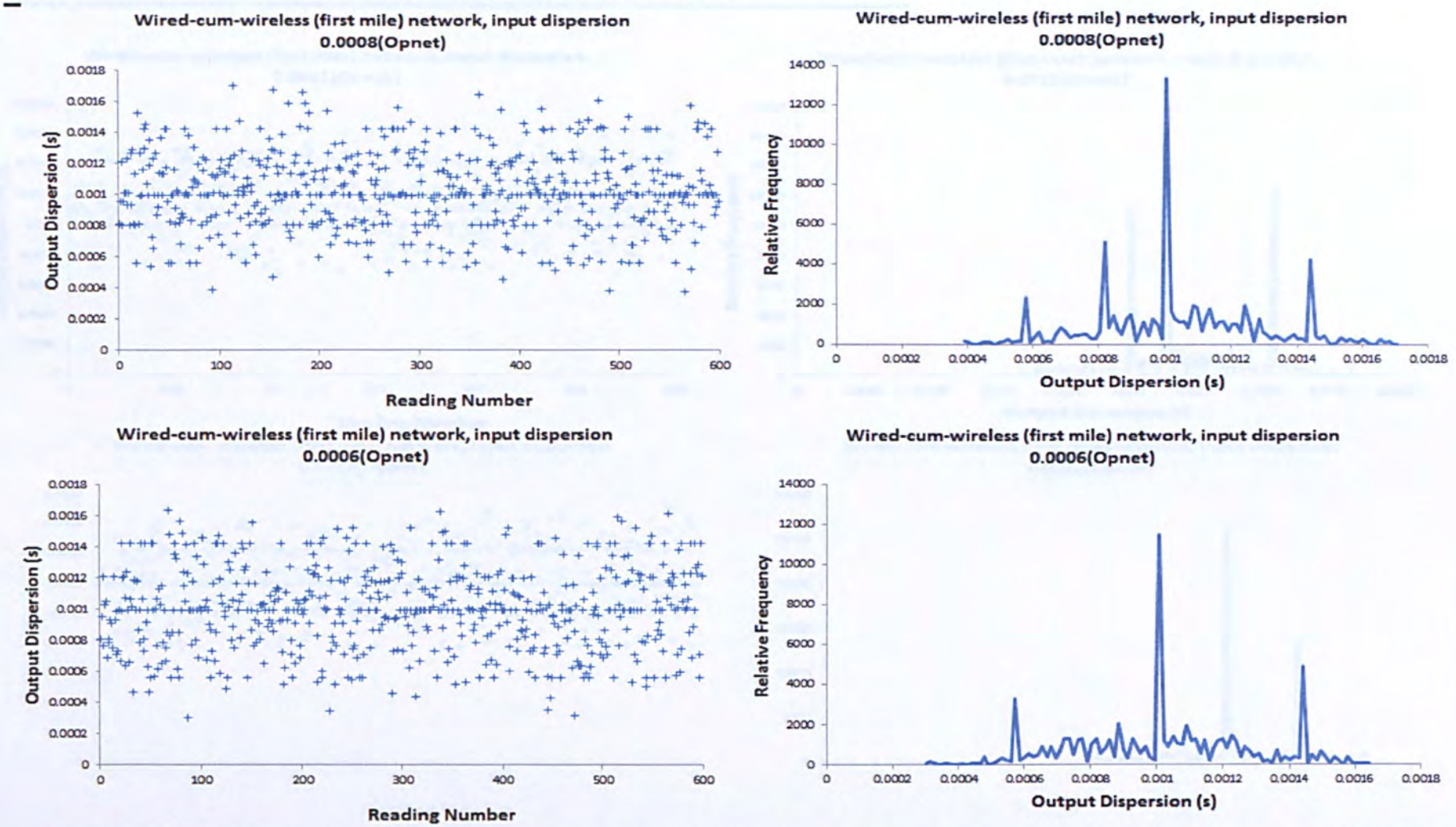


Figure 5.19 Raw dispersion data and histograms for wired-cum-wireless network using 128 byte probe packet pairs with input dispersions for 0.0008 and 0.0006 seconds.

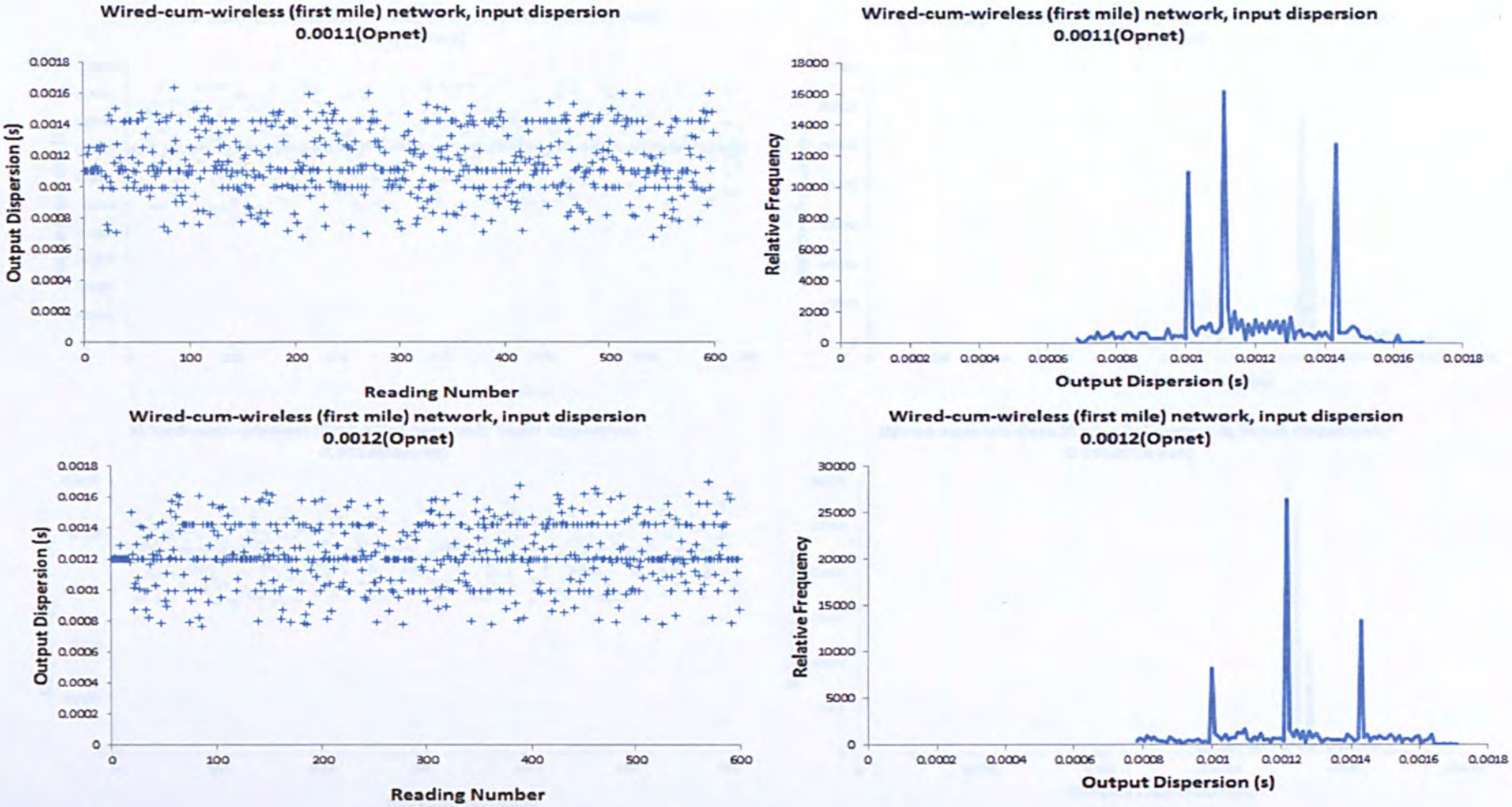


Figure 5.20 Raw dispersion data and histograms for wired-cum-wireless network using 128 byte probe packet pairs with input dispersions for 0.0011 and 0.0012 seconds.

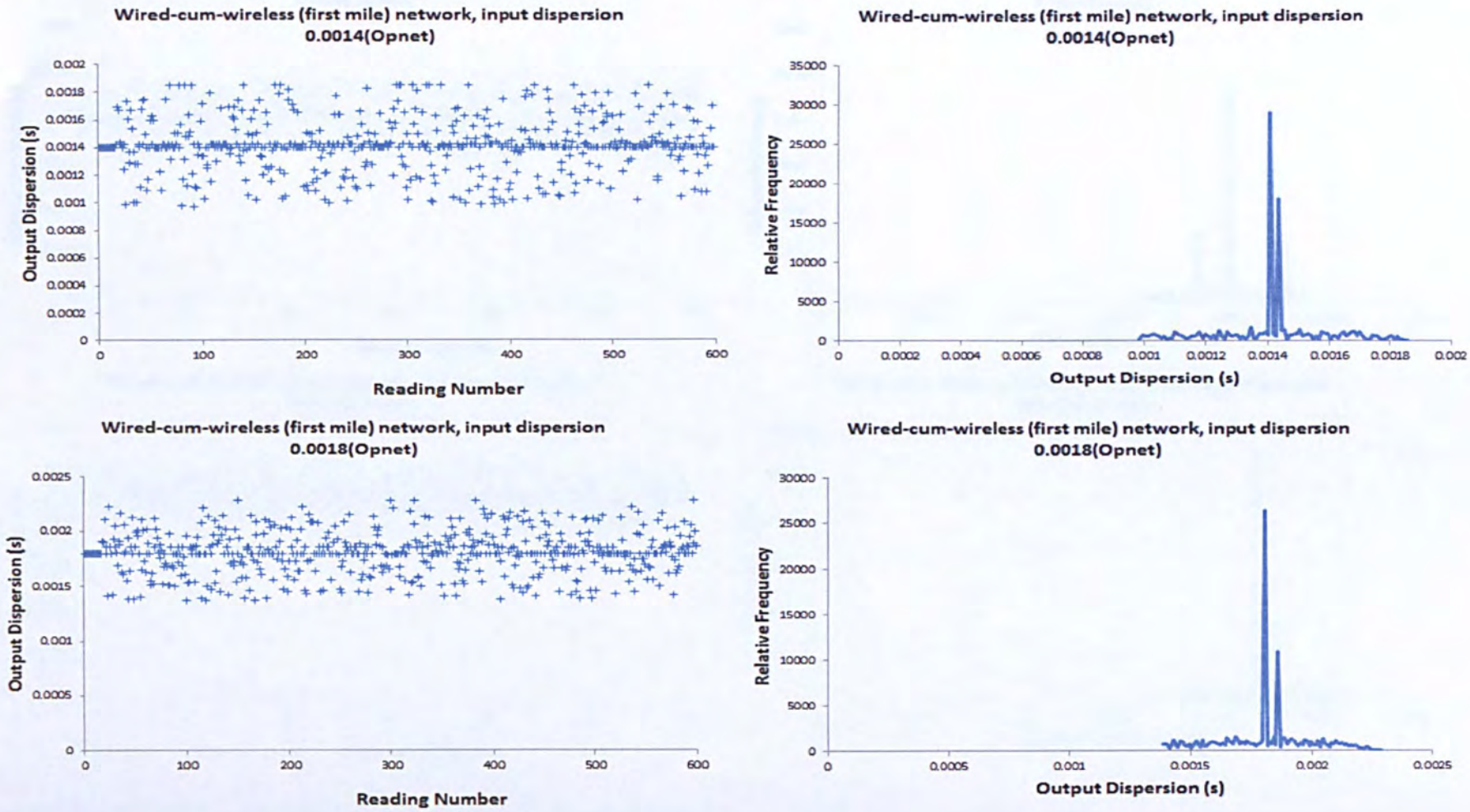


Figure 5.21 Raw dispersion data and histograms for wired-cum-wireless network using 128 byte probe packet pairs with input dispersions for 0.0014 and 0.0018 seconds

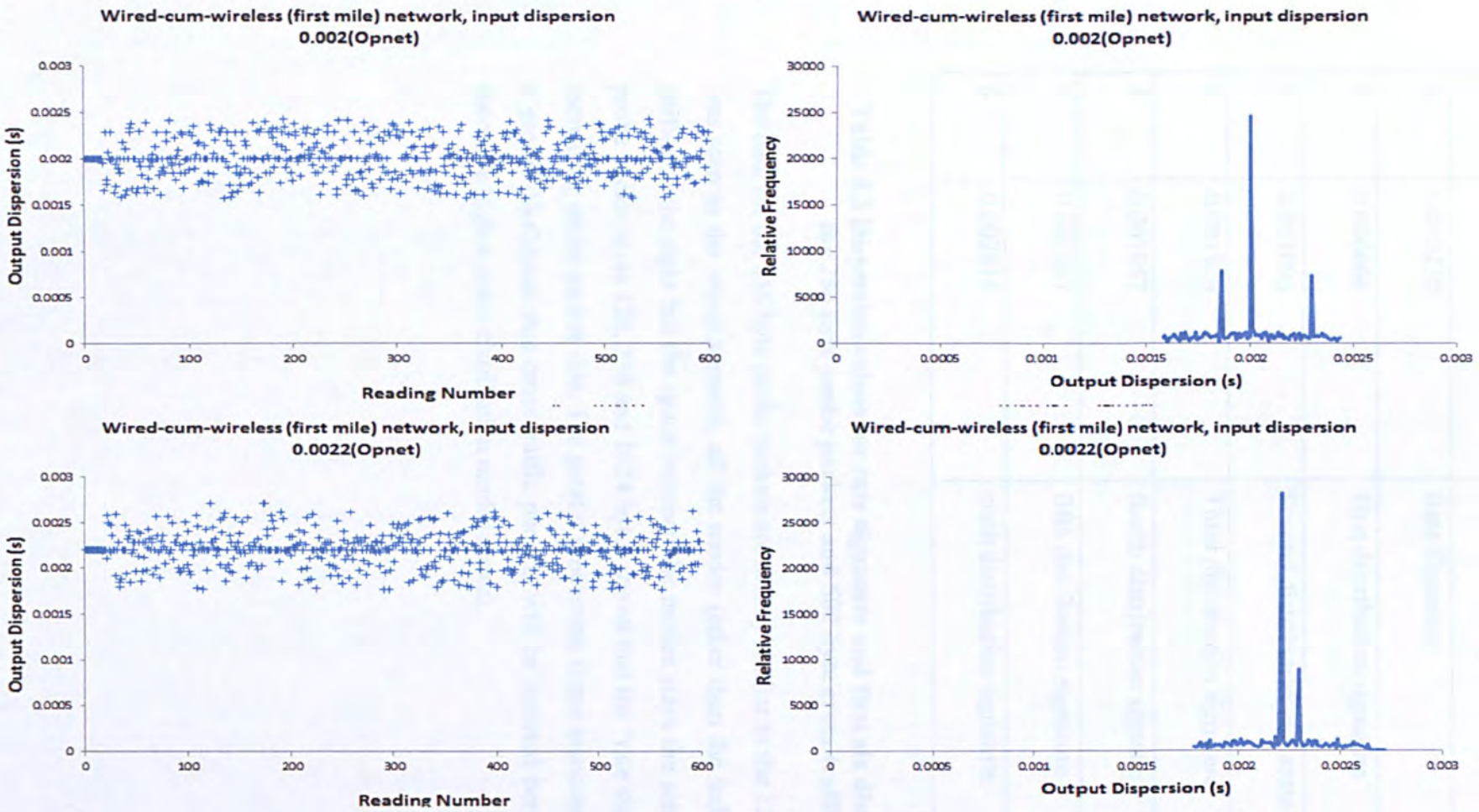


Figure 5.22 Raw dispersion data and histograms for wired-cum-wireless network using 128 byte probe packet pairs with input dispersions for 0.002 and 0.0022 seconds

n	Output Dispersion (s)	
0	0.000235	Rate Signature
1	0.000666	First distribution signature
2	0.001096	Second distribution signature
3	0.001526	Third distribution signature
4	0.001957	fourth distribution signature
5	0.002387	fifth distribution signature
6	0.002818	sixth distribution signature

Table 5.2 Dispersion values for rate signature and first six distribution signatures for 256 byte probe packets and 500 byte cross-traffic packets

The data for the 256 byte probe packets are mostly similar to the 128 byte probe packet data. As was seen in the wired scenario, all the modes (other than the independence mode) have been shifted to the right but the space between the modes stays the same. Observing the results for probe packet sizes 128, 256 and 1024 bytes shows that the "rate signature" mode increases with increasing probe packet size. The greater processing times associated with larger packets mean a greater likelihood that cross-traffic packets will be inserted between the probe packets, and therefore higher order distribution modes appear.

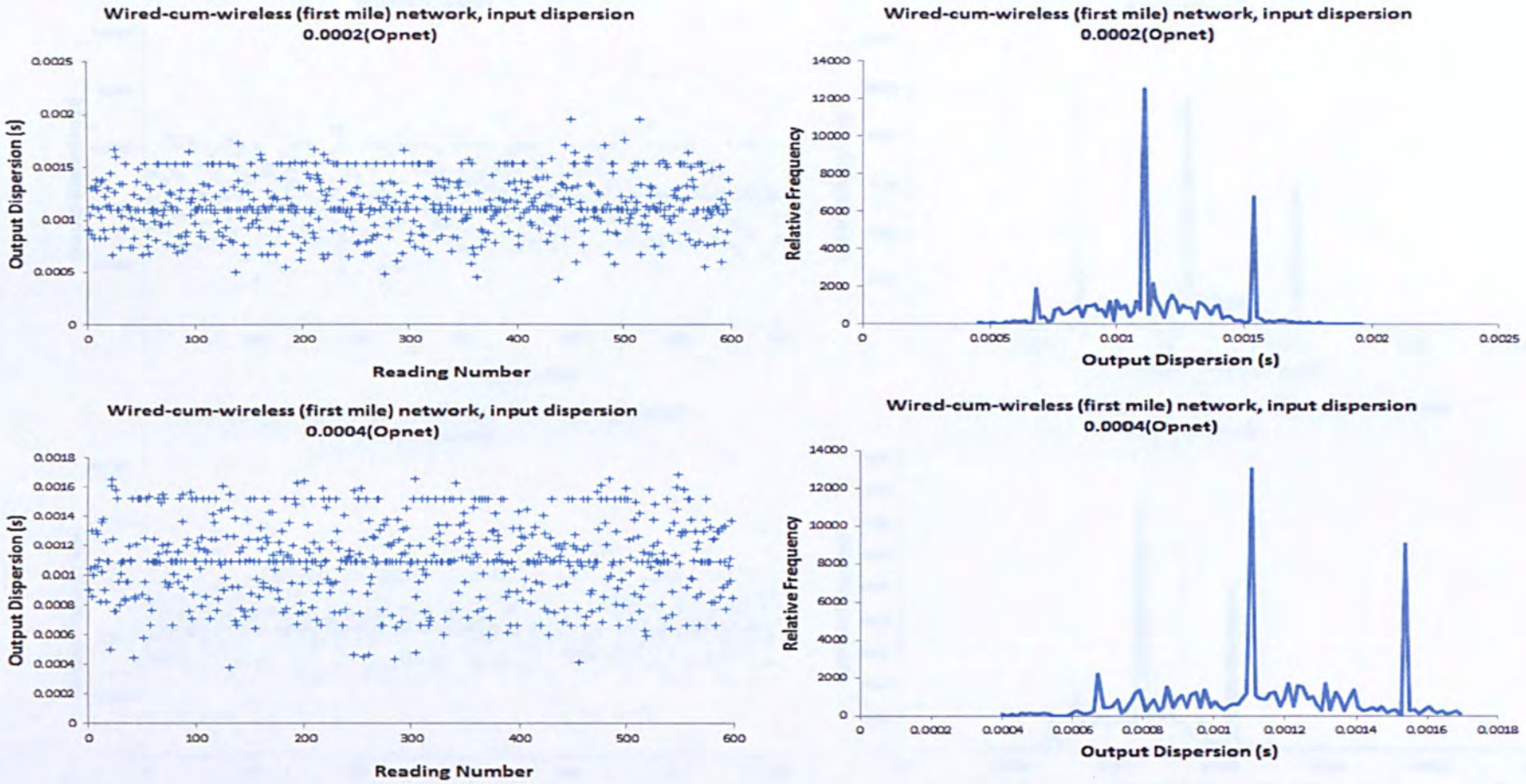


Figure 5.23 Raw dispersion data and histograms for wired-cum-wireless network using 256 byte probe packet pairs with input dispersions for 0.0002 and 0.0004 seconds.

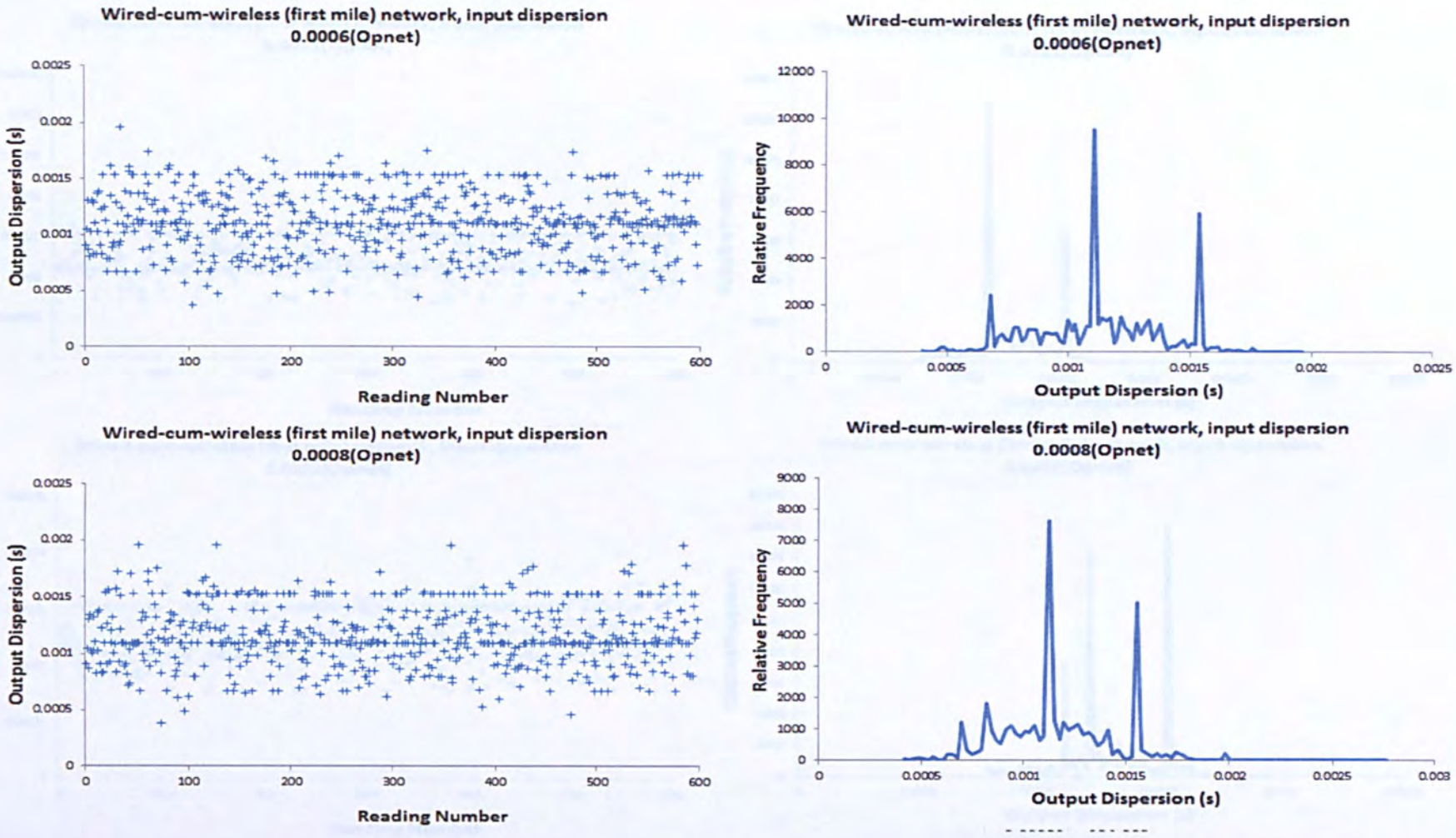


Figure 5.24 Raw dispersion data and histograms for wired-cum-wireless network using 256 byte probe packet pairs with input dispersions for 0.0006 and 0.0008 seconds.

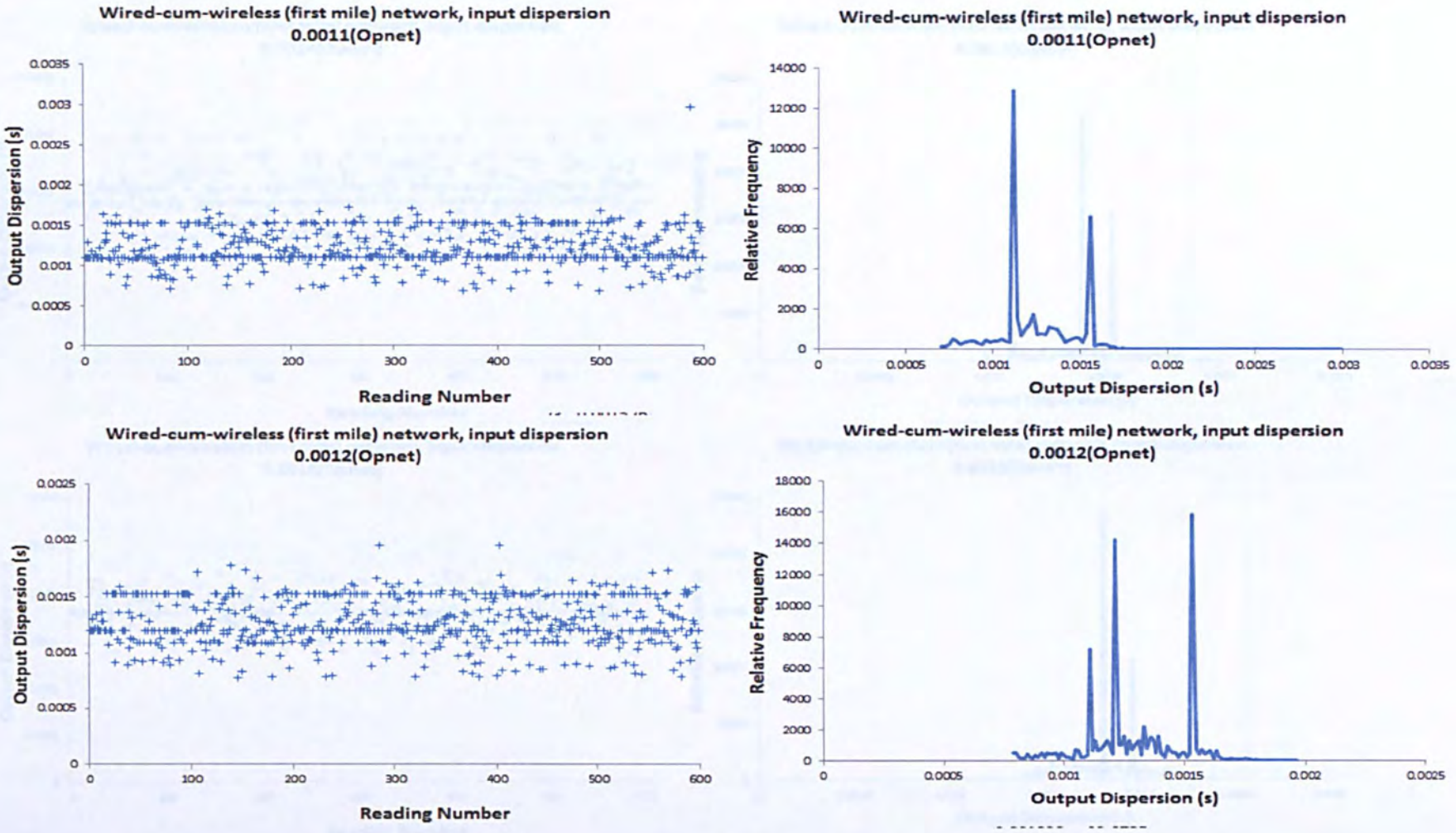


Figure 5.25 Raw dispersion data and histograms for wired-cum-wireless network using 256 byte probe packet pairs with input dispersions for 0.0011 and 0.0012 seconds.

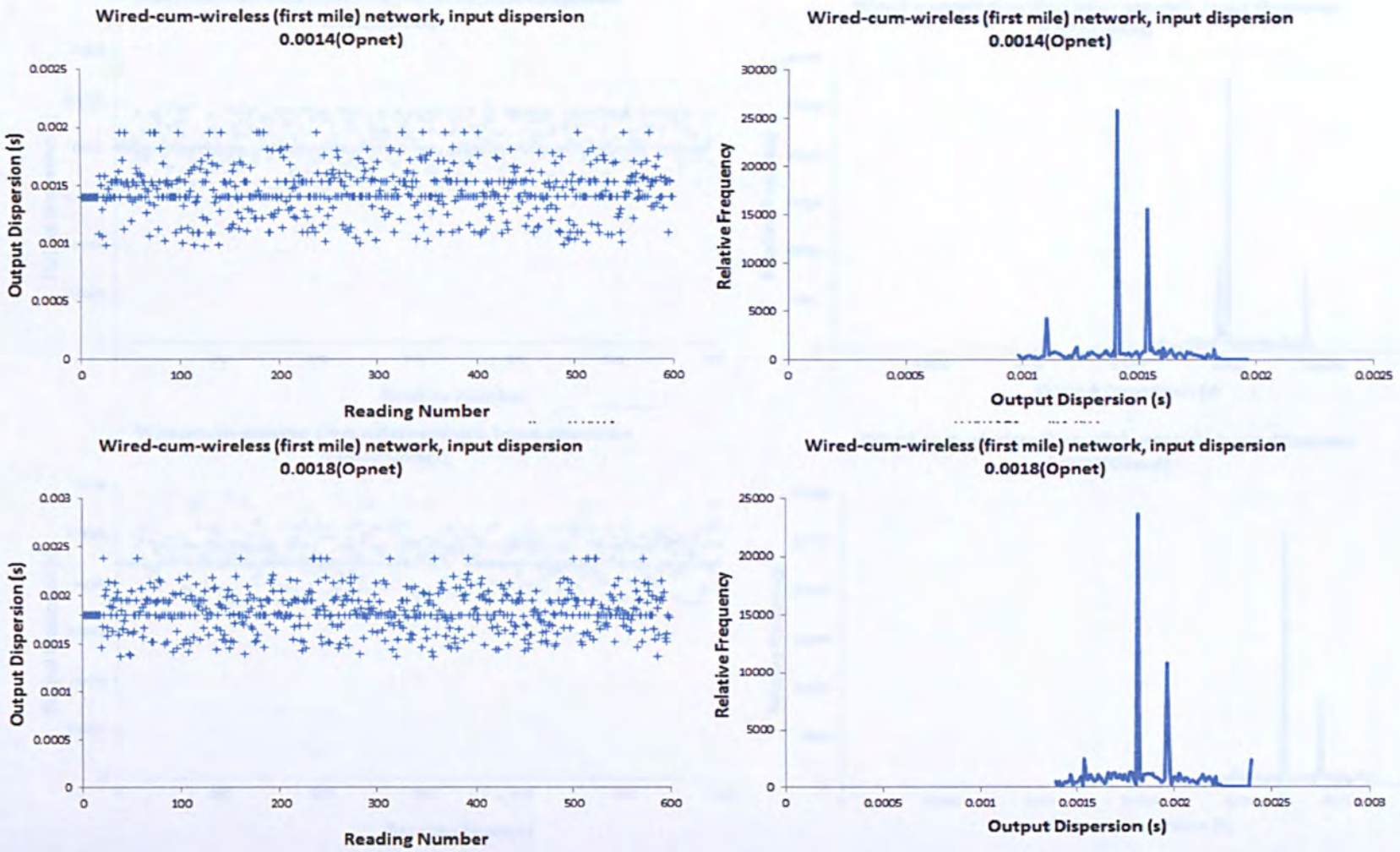


Figure 5.26 Raw dispersion data and histograms for wired-cum-wireless network using 256 byte probe packet pairs with input dispersions for 0.0014 and 0.0018 seconds.

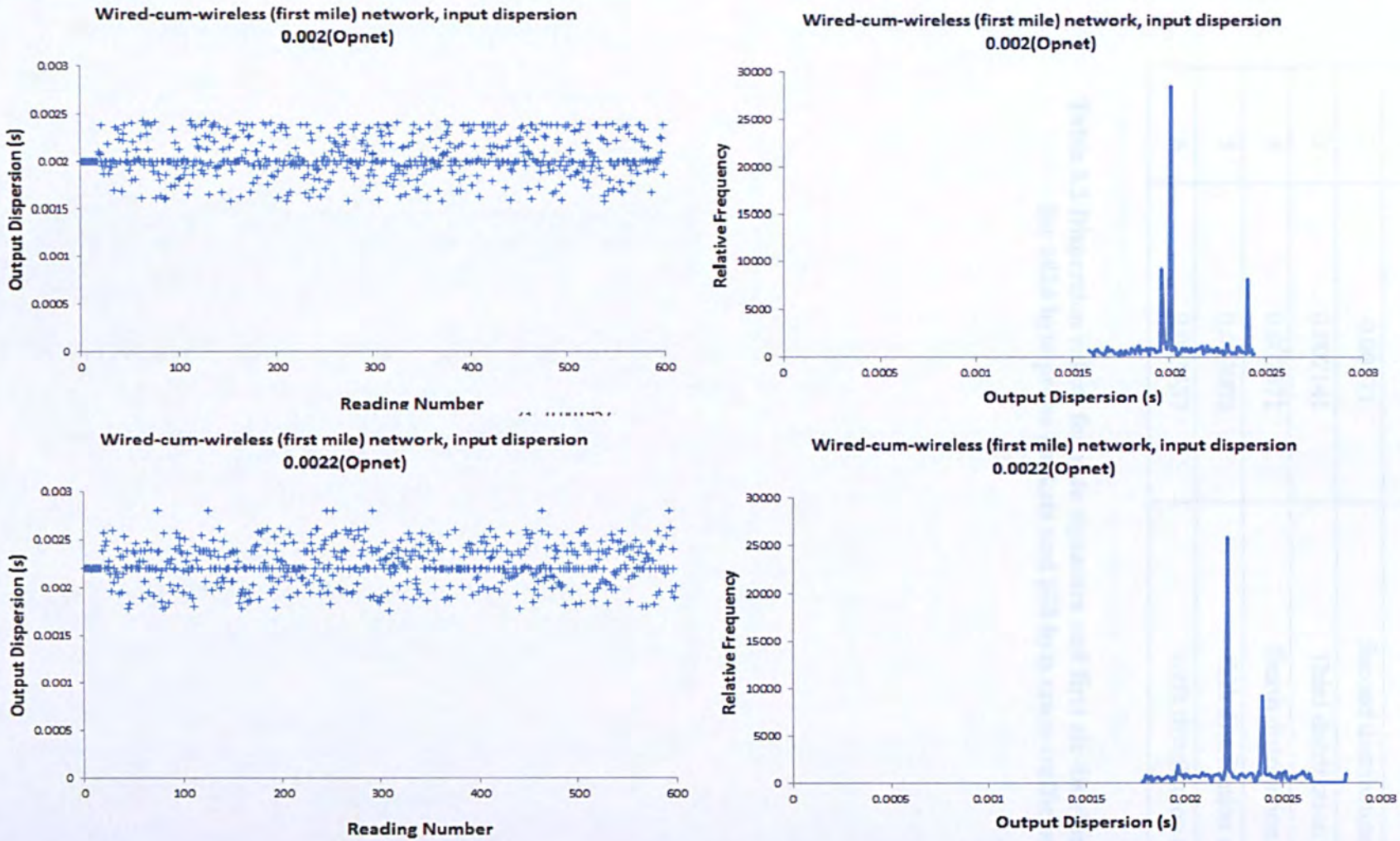


Figure 5.27 Raw dispersion data and histograms for wired-cum-wireless network using 256 byte probe packet pairs with input dispersions for 0.002 and 0.0022 seconds.

n	**Output Dispersion (s)**	
0	0.00085	Rate Signature
1	0.00128	First distribution signature
2	0.00171	Second distribution signature
3	0.002141	Third distribution signature
4	0.002571	fourth distribution signature
5	0.003002	fifth distribution signature
6	0.003432	sixth distribution signature

Table 5.3 Dispersion values for rate signature and first six distribution signatures for 1024 byte probe packets and 500 byte cross-traffic packets

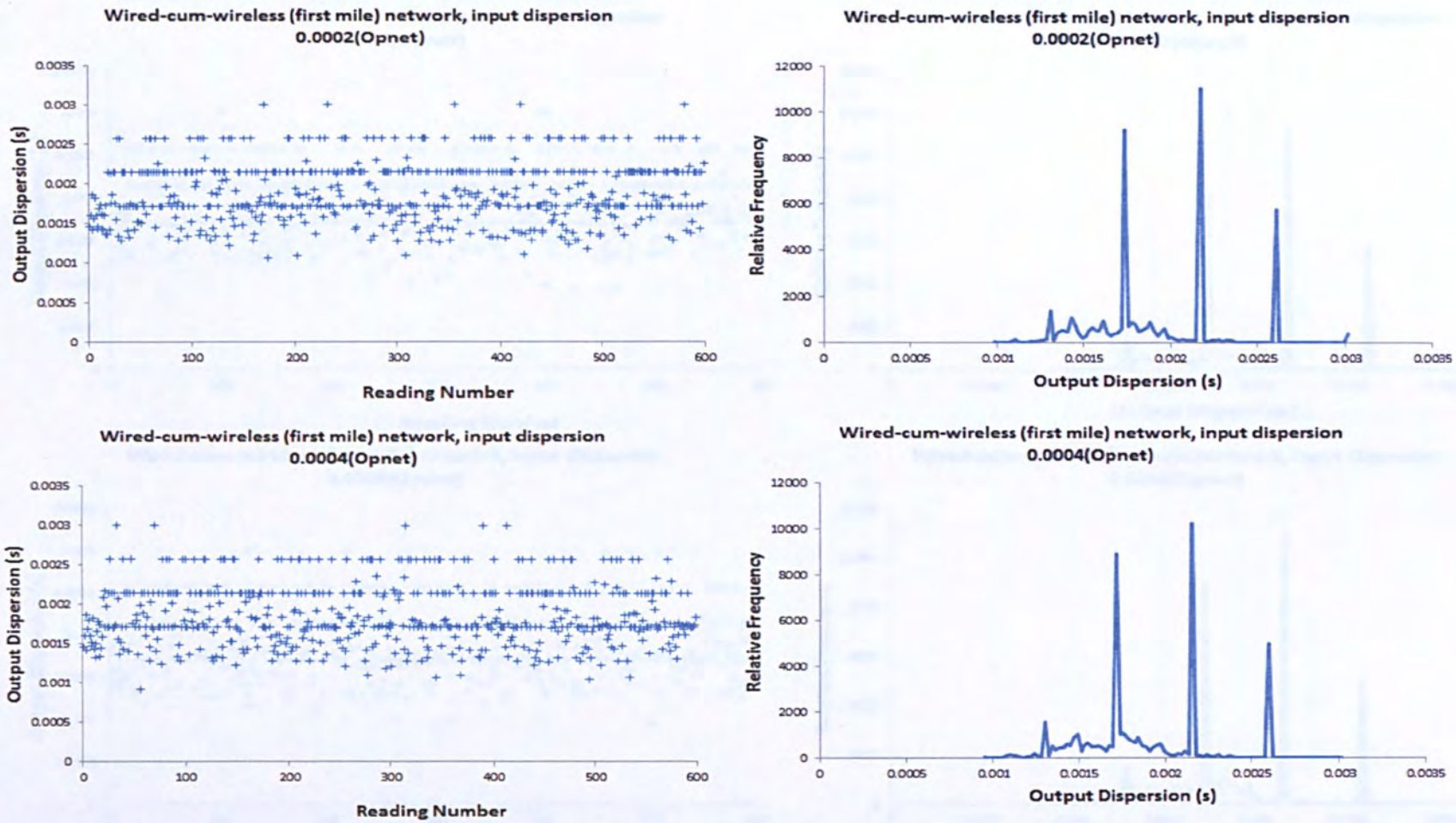


Figure 5.28 Raw dispersion data and histograms for wired-cum-wireless network using 1024 byte probe packet pairs with input dispersions for 0.0002 and 0.0004 seconds.

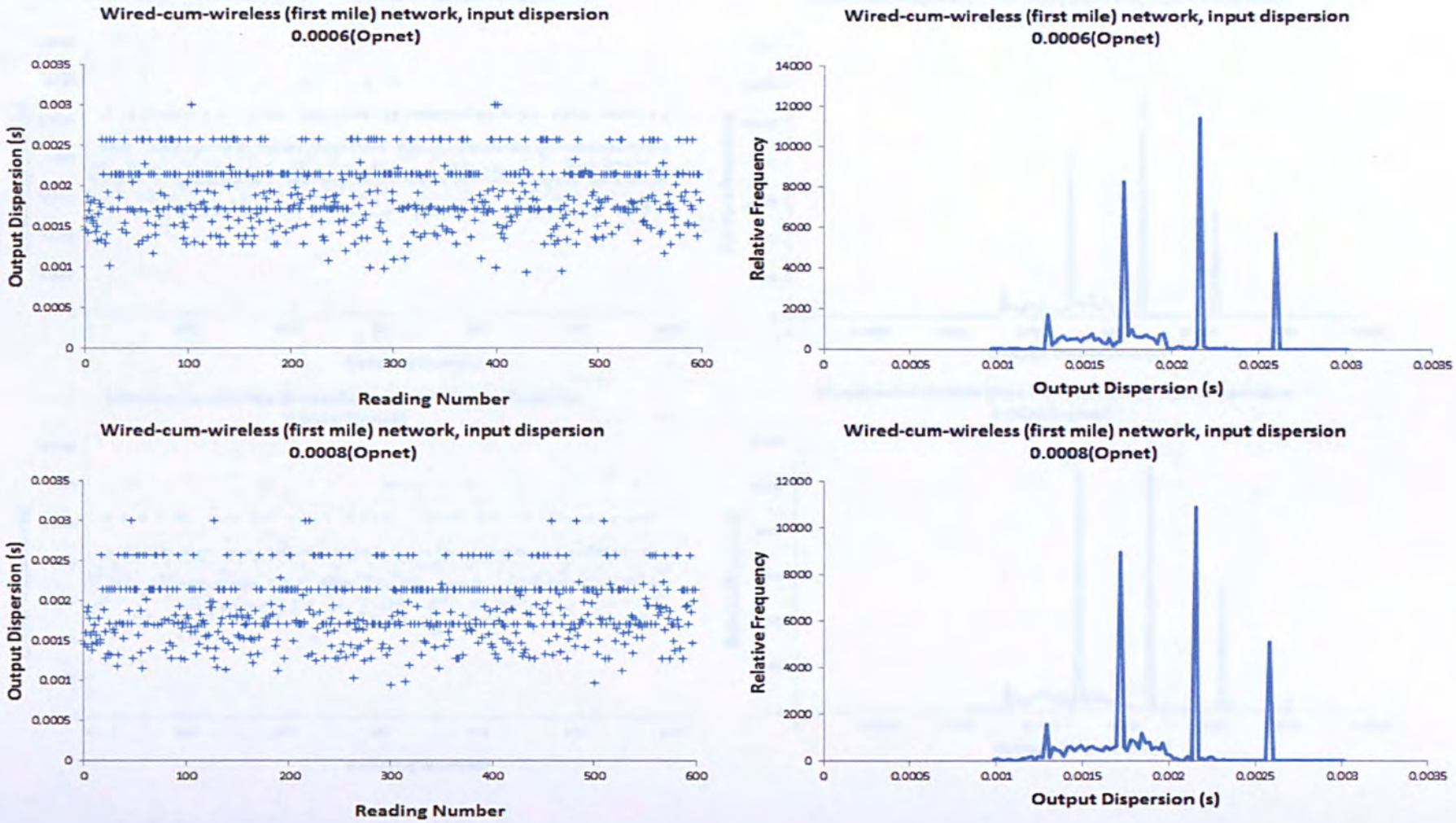


Figure 5.29 Raw dispersion data and histograms for wired-cum-wireless network using 1024 byte probe packet pairs with input dispersions for 0.0004 and 0.0008 seconds.

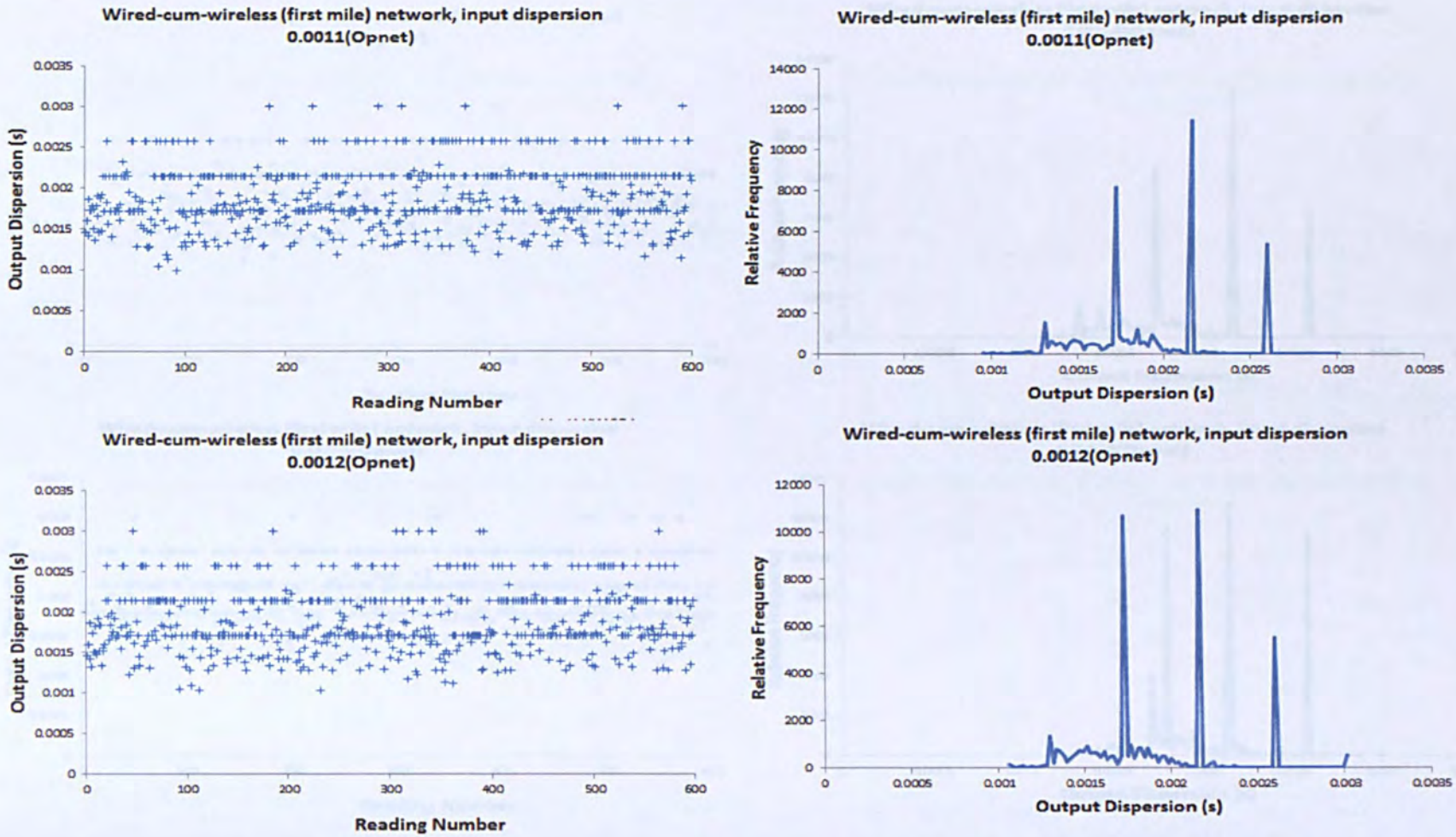


Figure 5.30 Raw dispersion data and histograms for wired-cum-wireless network using 1024 byte probe packet pairs with input dispersions for 0.0011 and 0.0012 seconds.

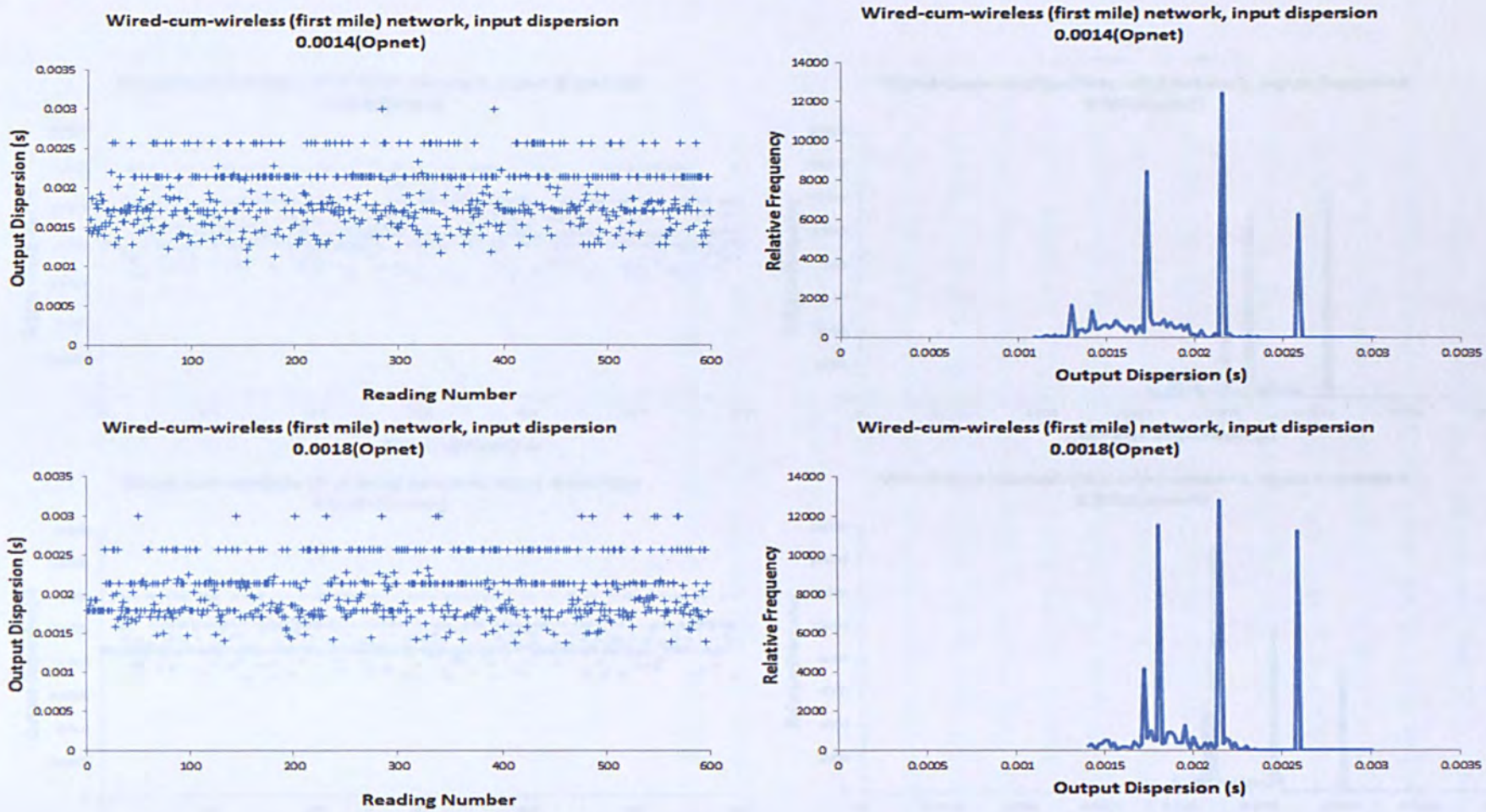


Figure 5.31 Raw dispersion data and histograms for wired-cum-wireless network using 1024 byte probe packet pairs with input dispersions for 0.0014 and 0.0018 seconds.

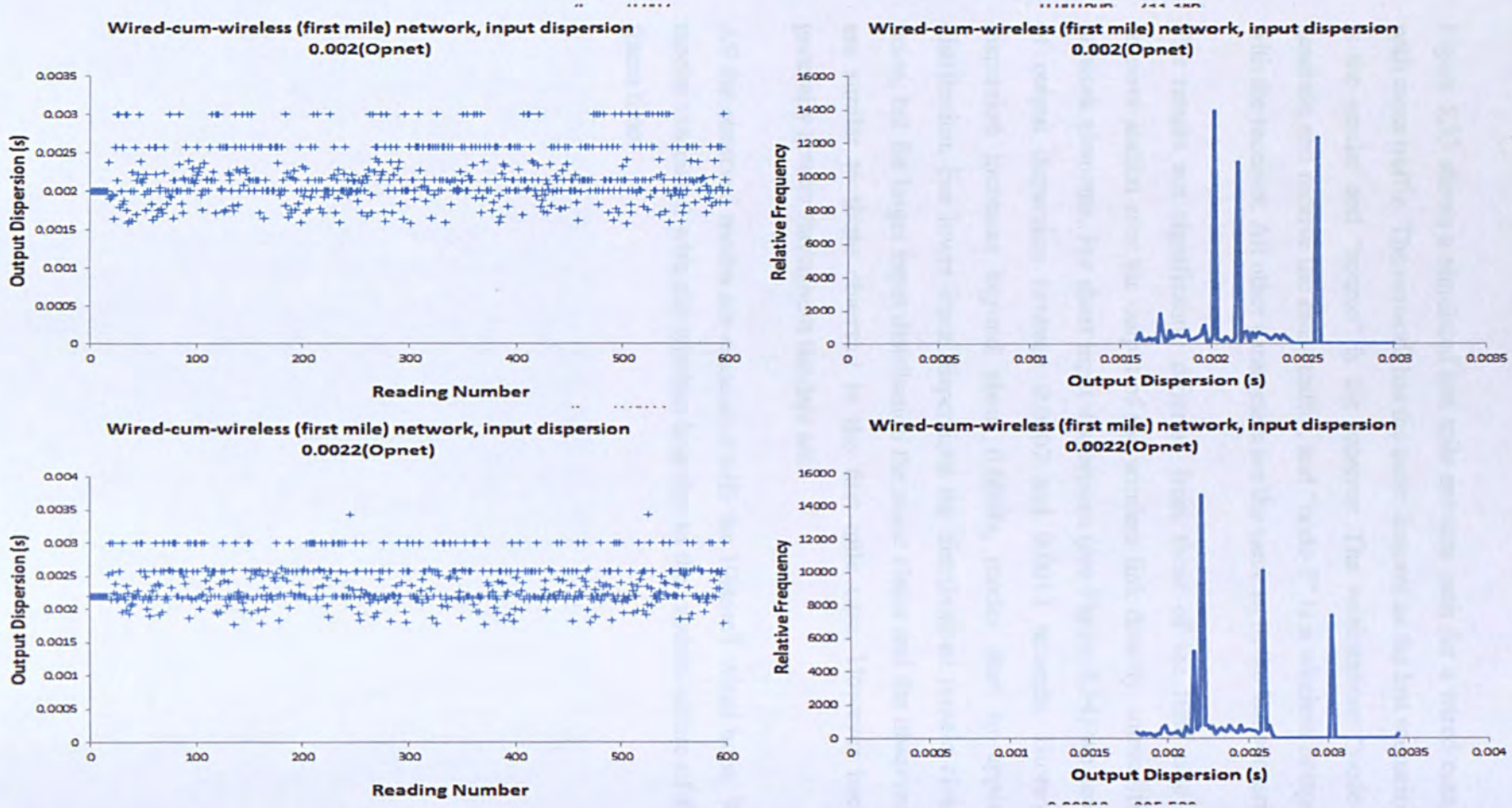


Figure 5.32 Raw dispersion data and histograms for wired-cum-wireless network using 1024 byte probe packet pairs with input dispersions for 0.002 and 0.0022 seconds.

5.3.3. Last Mile Simulated Topology

Figure 5.33 shows a simulated last mile network path for a wired-cum-wireless network with cross traffic. The network has the same features as the last scenario, but the "node-2" is the sender and "source" is the receiver. The workstations "node-3" and "node-4" generate and receive the cross traffic, and "node-7" is a wireless hotspot communicating with the receiver. All other parameters are the same as in the last scenario.

The results are significantly different from those of the first mile case because the receiver station sees the output of the wireless link directly, unmodified by subsequent network elements. For short input dispersions (see Figure 5.34) we see a uniform spread of output dispersion between 0.0007 and 0.0013 seconds. However, as the input dispersion increases beyond about 0.0008s, modes start to appear in the output distribution. For lower input dispersions the distributions remain clouded with random noise, but for larger input distribution the noise clears and the observed signature modes are similar to those observed in the first mile case. However background noise is generally greater throughout the data set.

All the observed modes are associated with the 10BaseT wired hops. We see no discrete modes associated with the wireless hop due to the random nature of the effective inter-frame time.

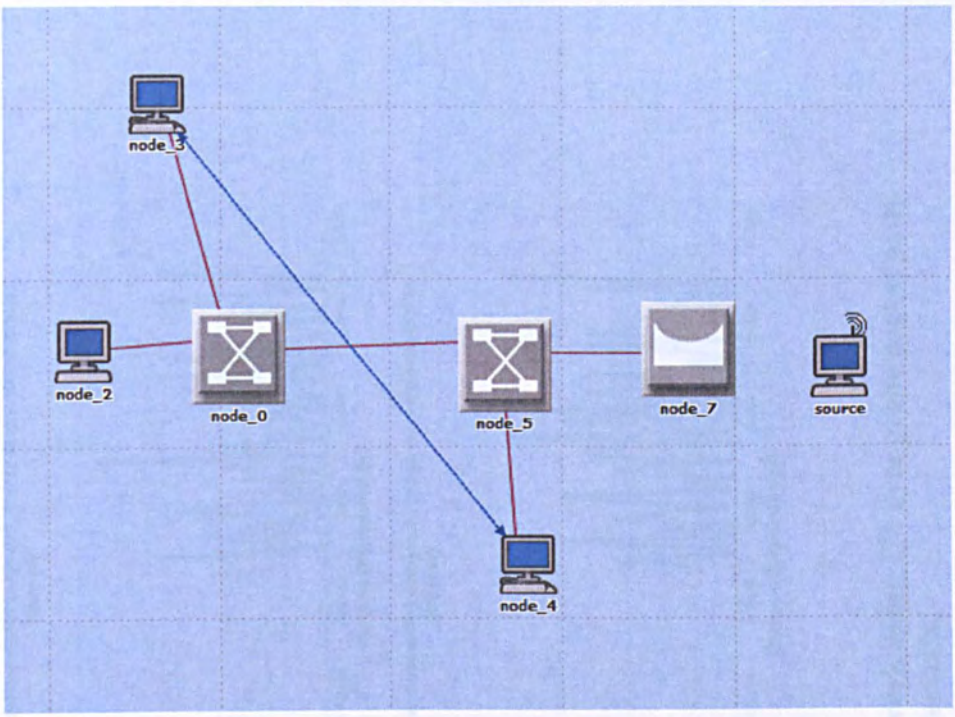


Figure 5.33 Wired-cum-Wireless network with Poisson cross-traffic, simulated in Opnet

5.3.4. Last Mile Results

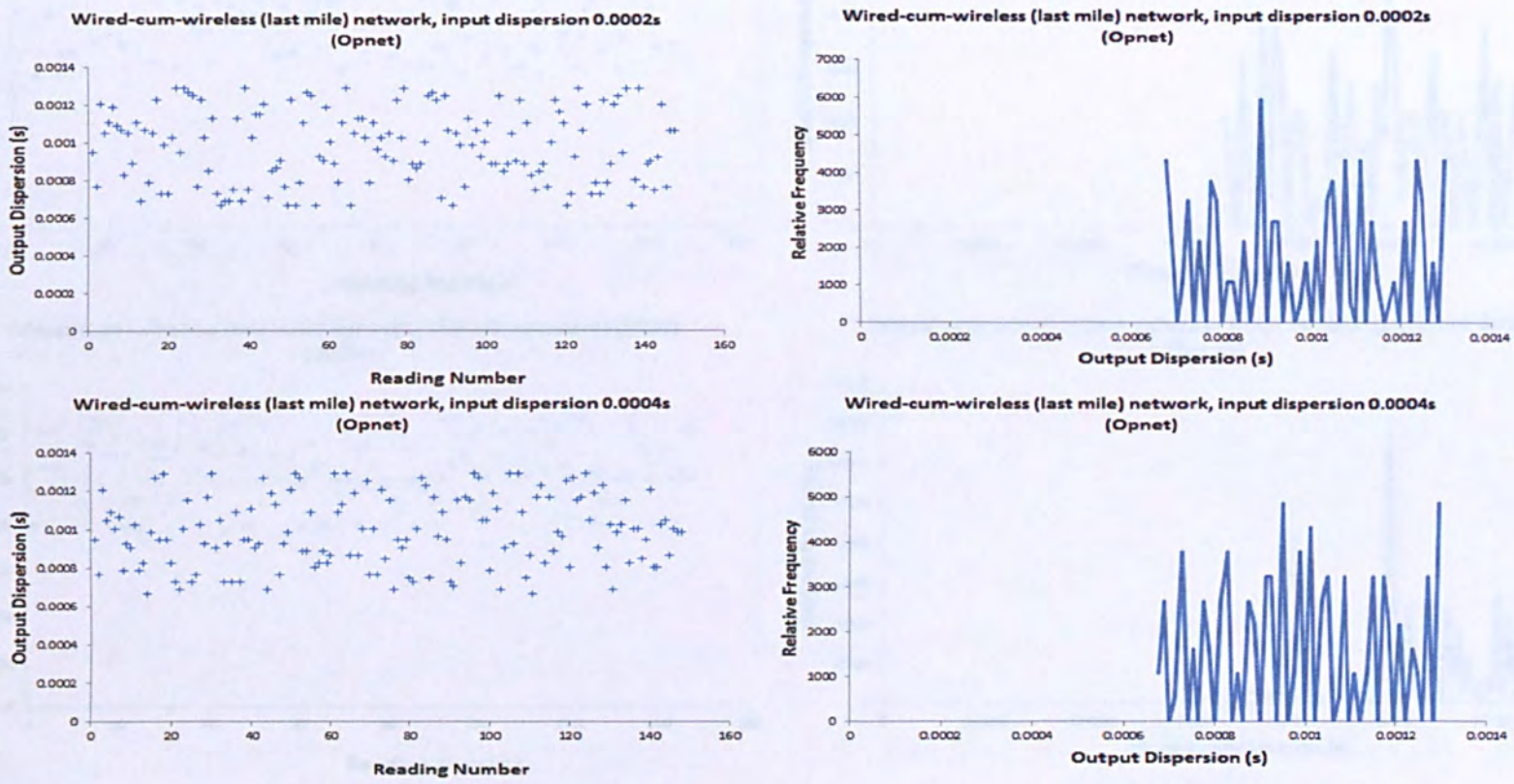


Figure 5.34 Raw dispersion data and histograms for wired-cum-wireless network using 128 byte probe packet pairs with input dispersions for 0.0002 and 0.0004 seconds.

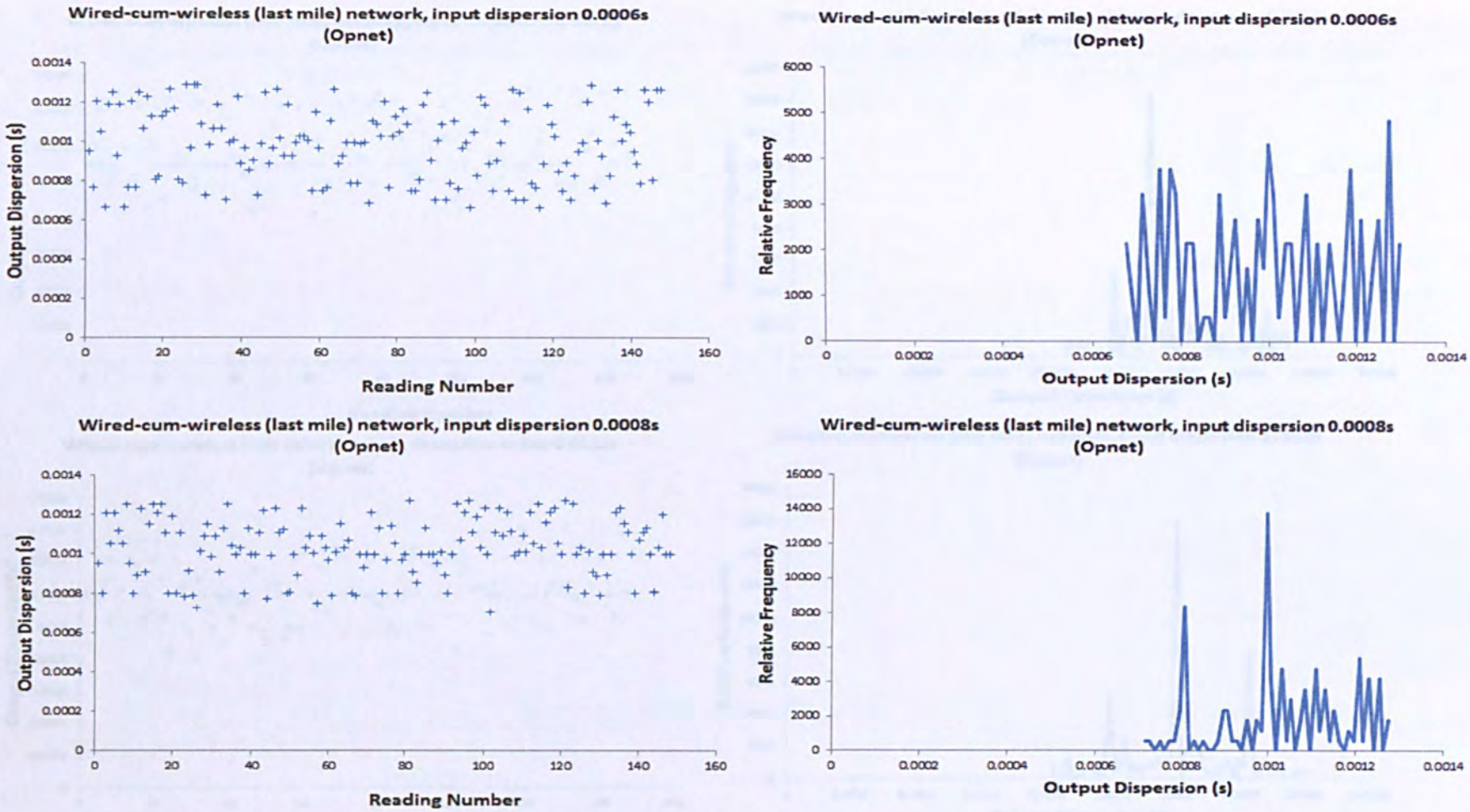


Figure 5.35 Raw dispersion data and histograms for wired-cum-wireless network using 128 byte probe packet pairs with input dispersions for 0.0006 and 0.0008 seconds.

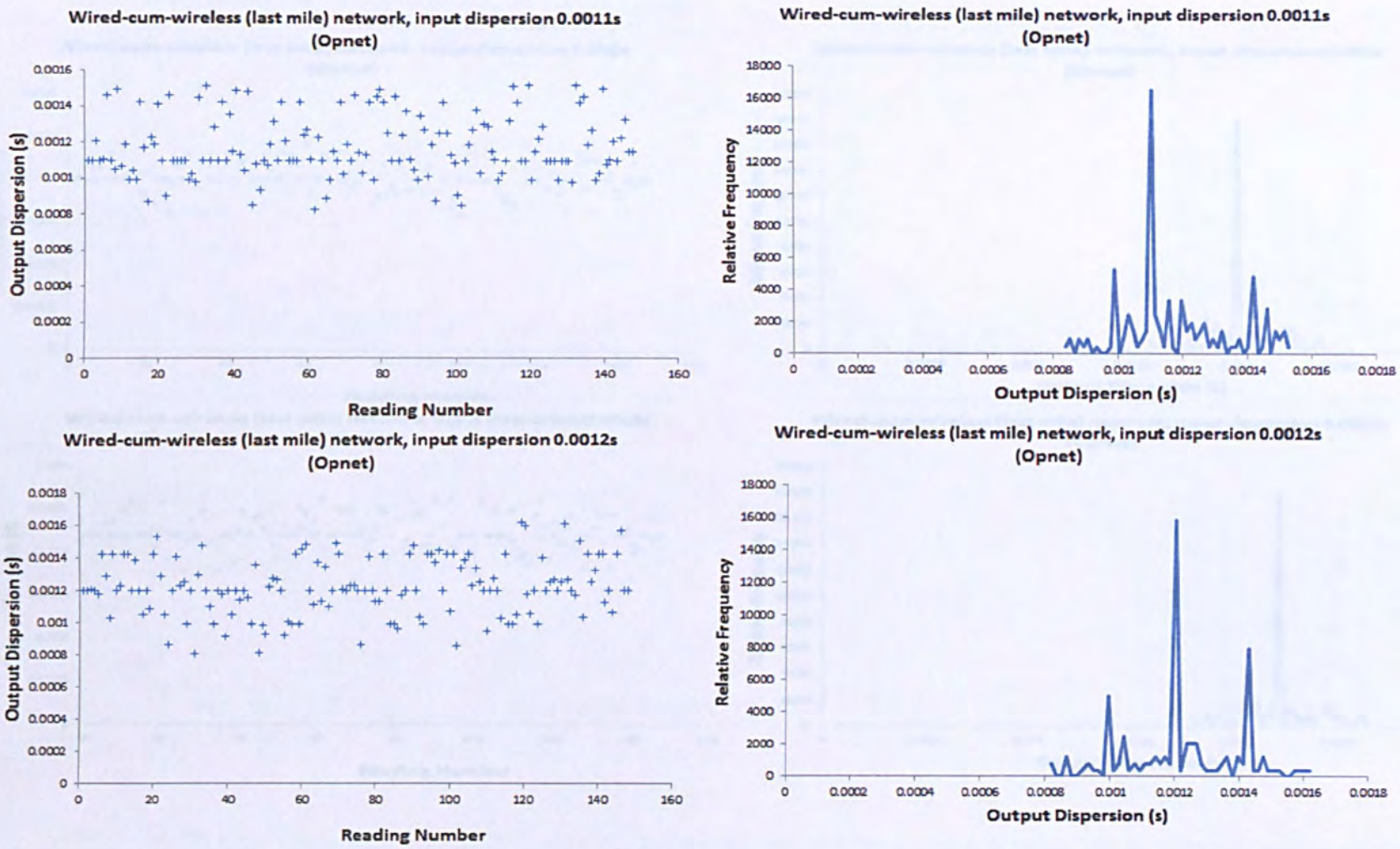


Figure 5.36 Raw dispersion data and histograms for wired-cum-wireless network using 128 byte probe packet pairs with input dispersions for 0.0011 and 0.0012 seconds.

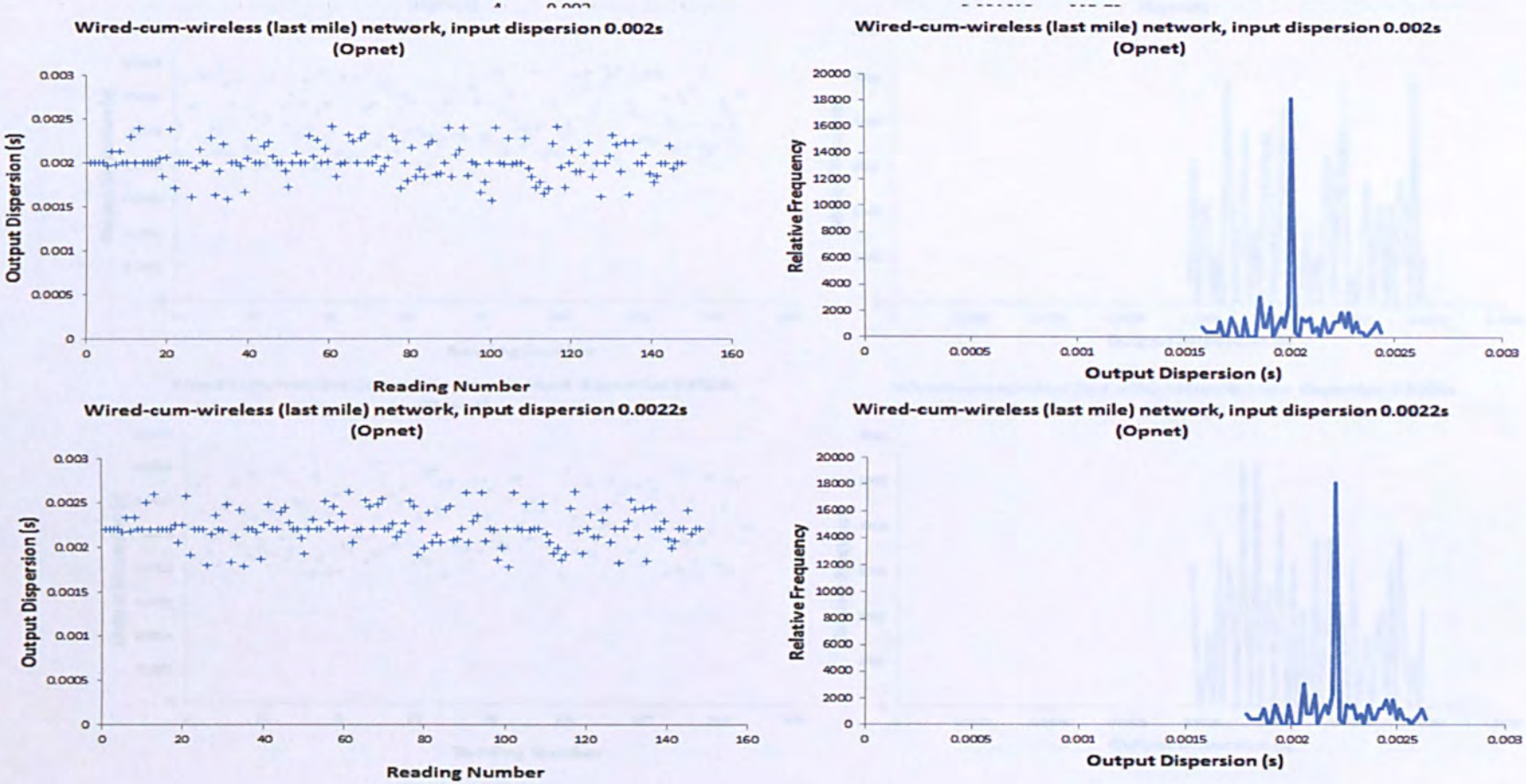


Figure 5.37 Raw dispersion data and histograms for wired-cum-wireless network using 128 byte probe packet pairs with input dispersions for 0.002 and 0.0022 seconds.

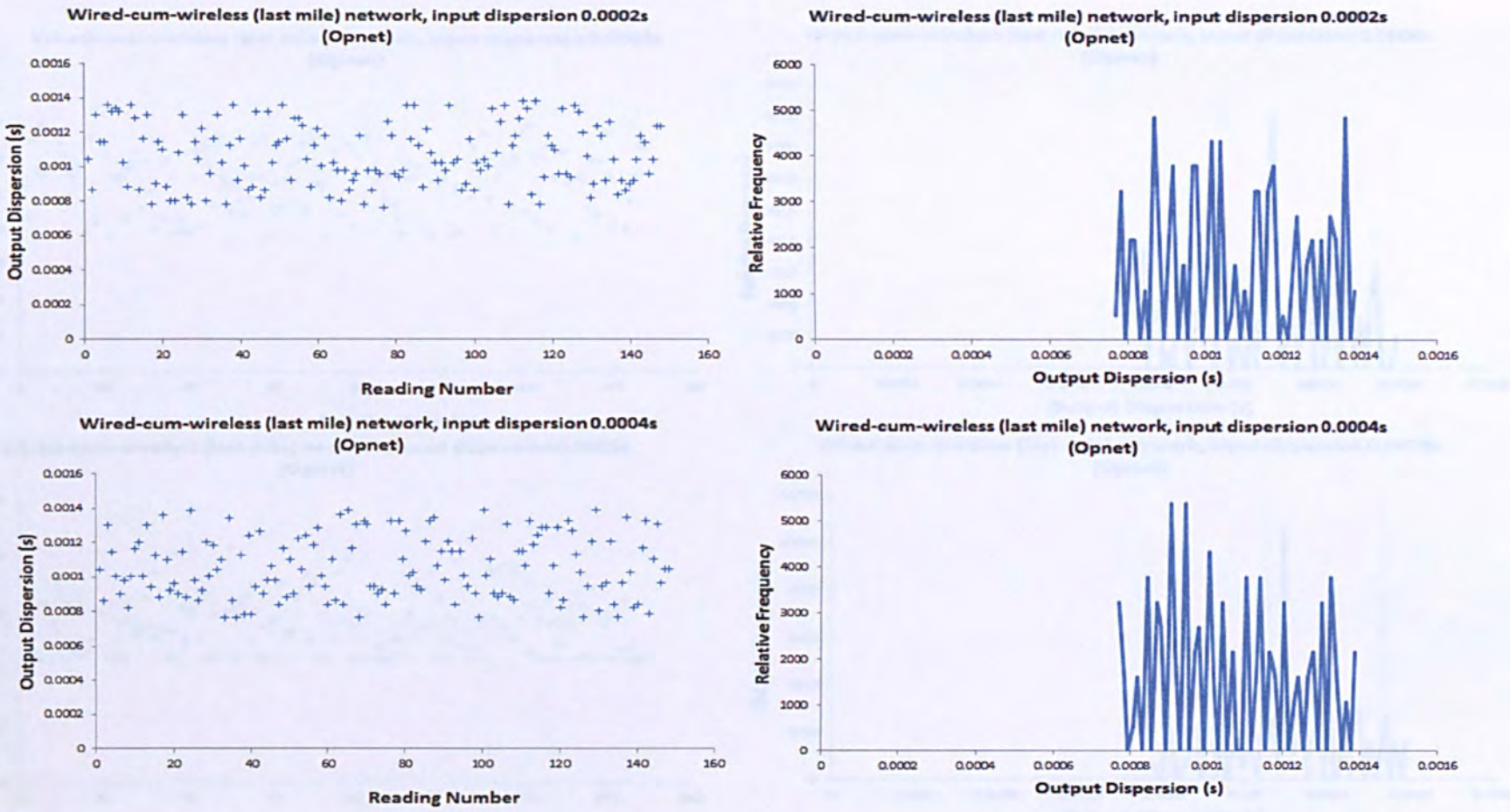


Figure 5.38 Raw dispersion data and histograms for wired-cum-wireless network using 256 byte probe packet pairs with input dispersions for 0.0002 and 0.0004 seconds.

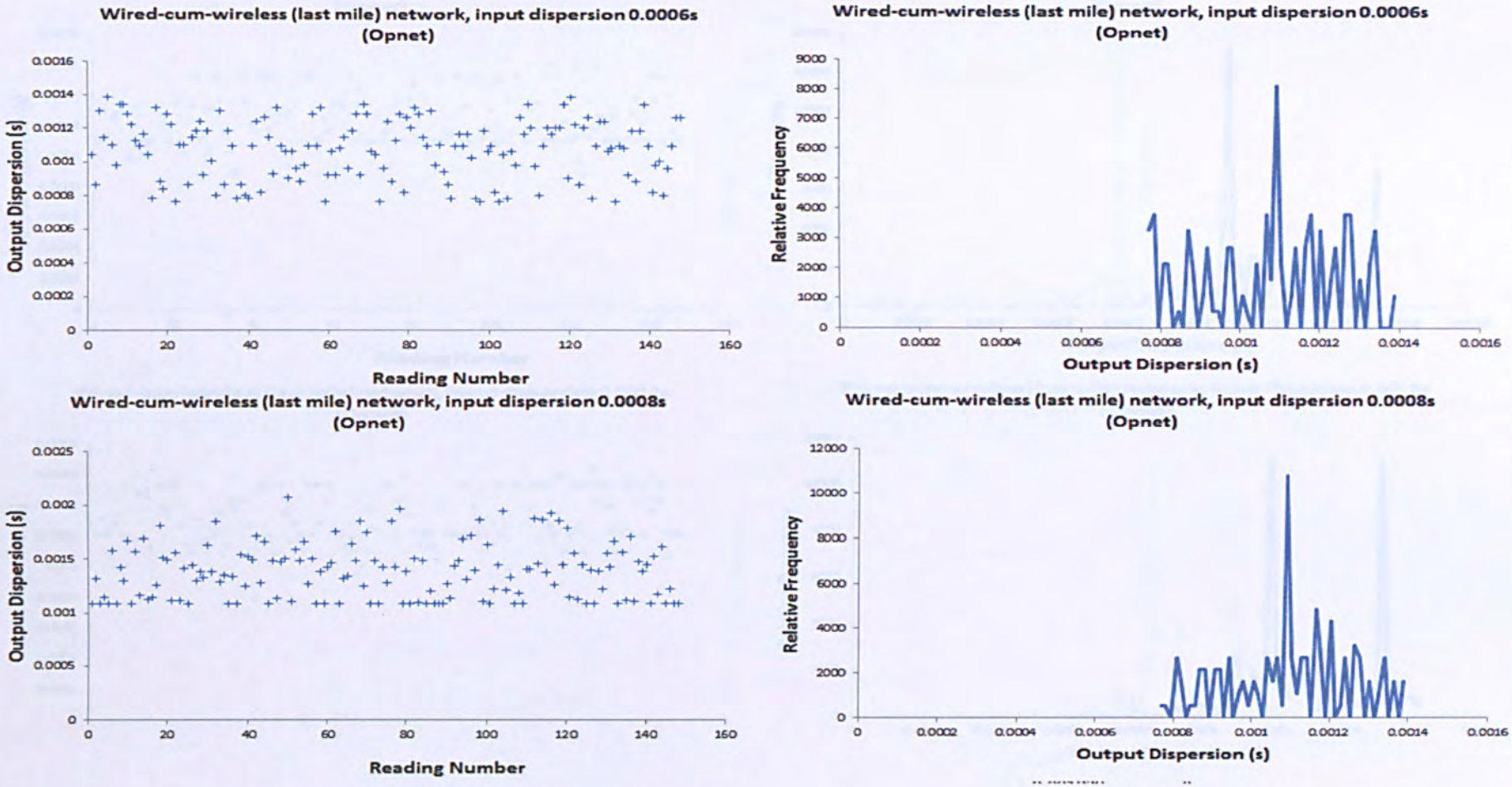


Figure 5.39 Raw dispersion data and histograms for wired-cum-wireless network using 256 byte probe packet pairs with input dispersions for 0.0006 and 0.0008 seconds.

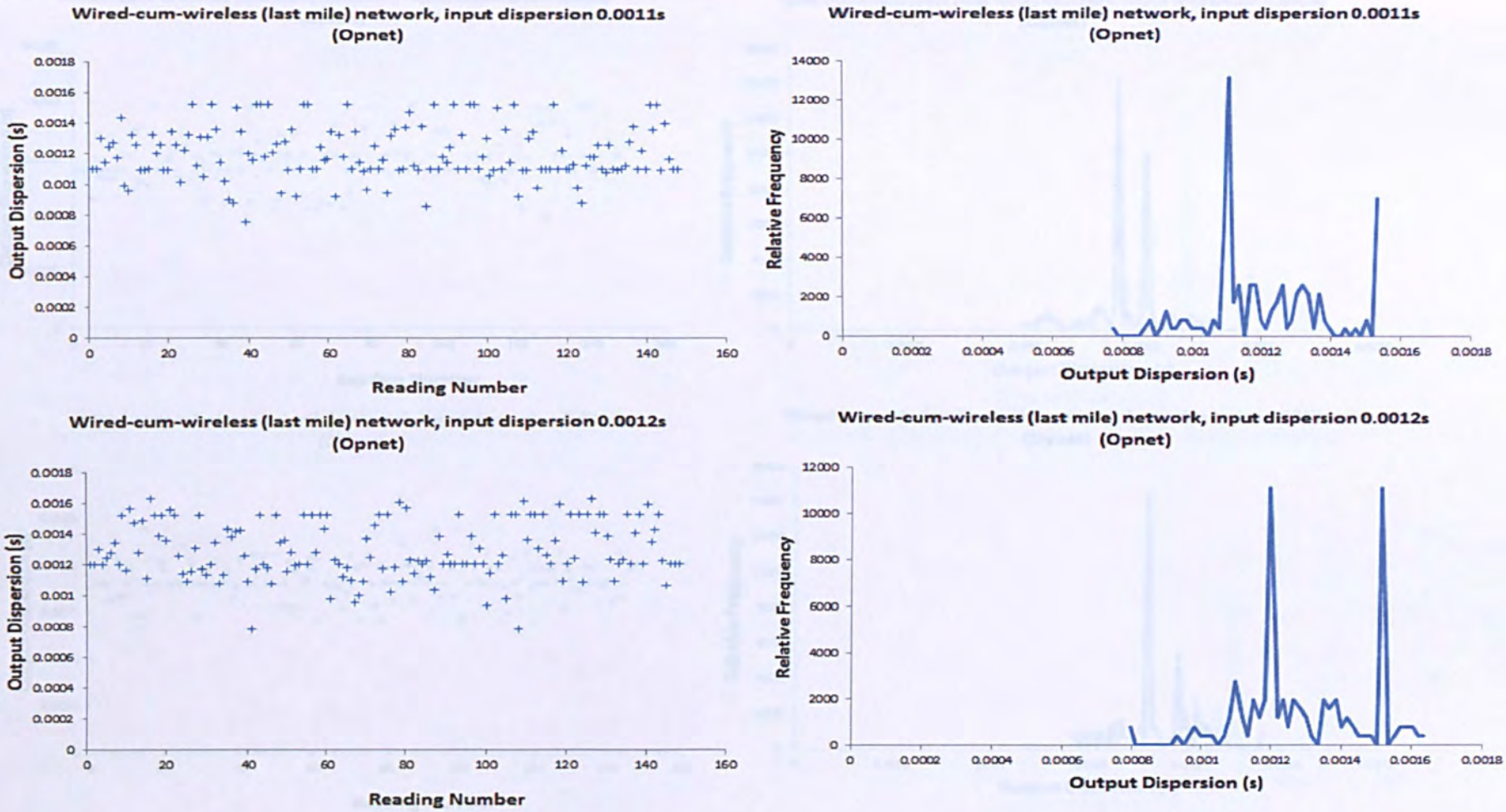


Figure 5.40 Raw dispersion data and histograms for wired-cum-wireless network using 256 byte probe packet pairs with input dispersions for 0.0011 and 0.0012 seconds.

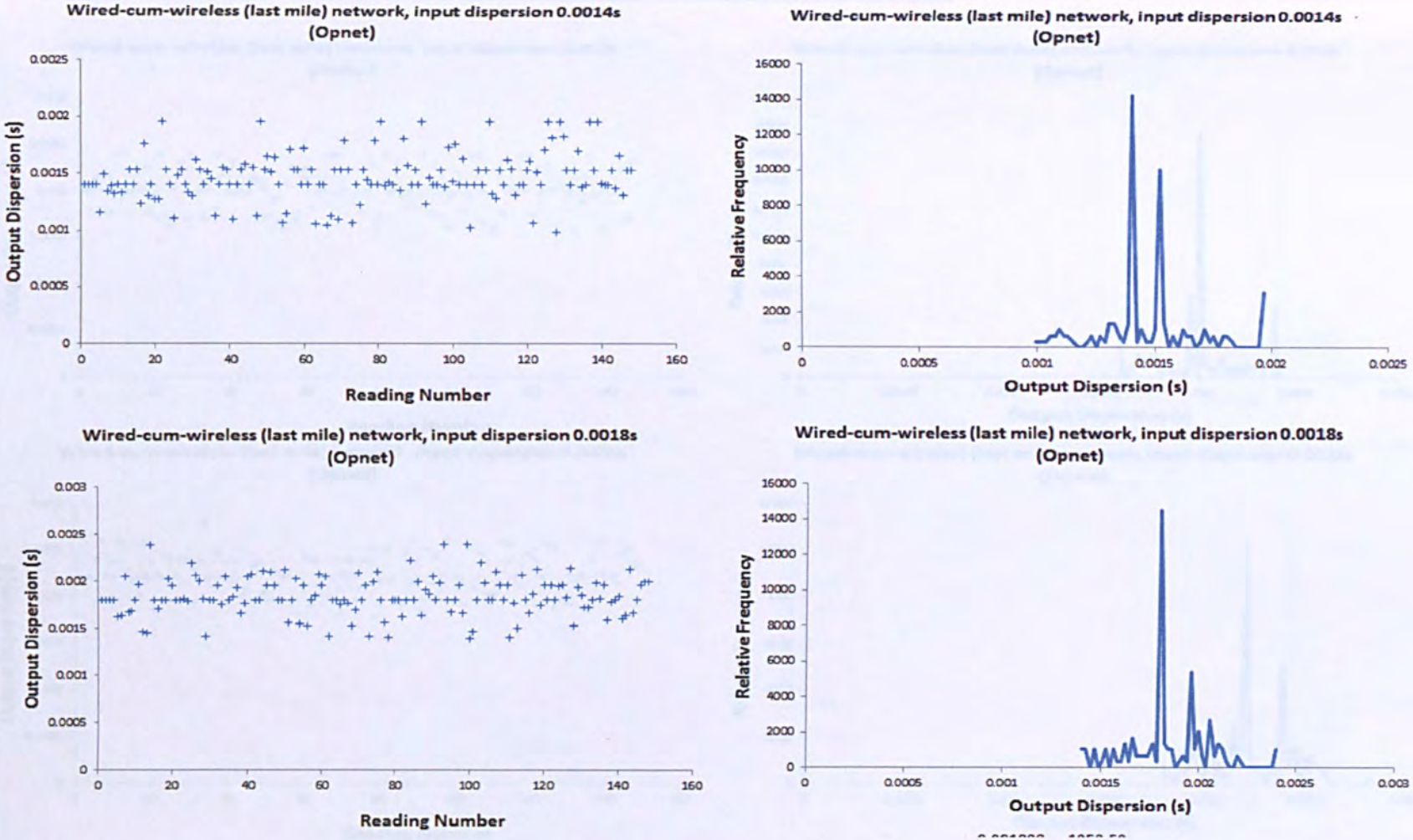
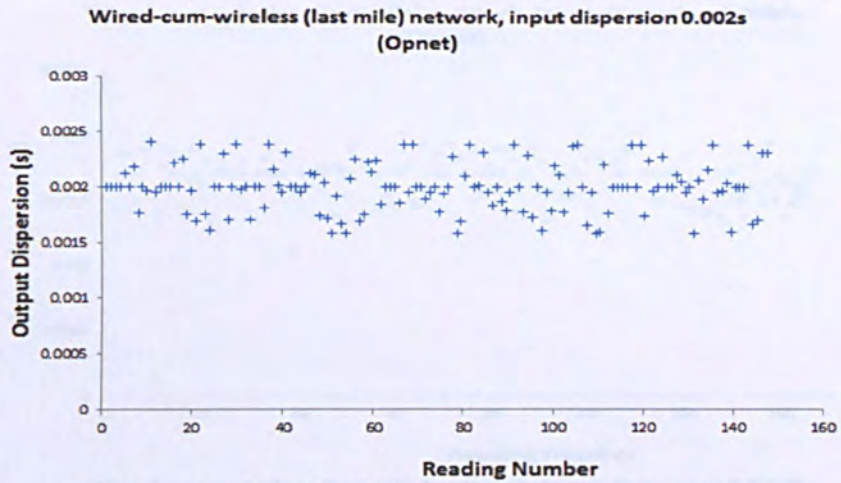
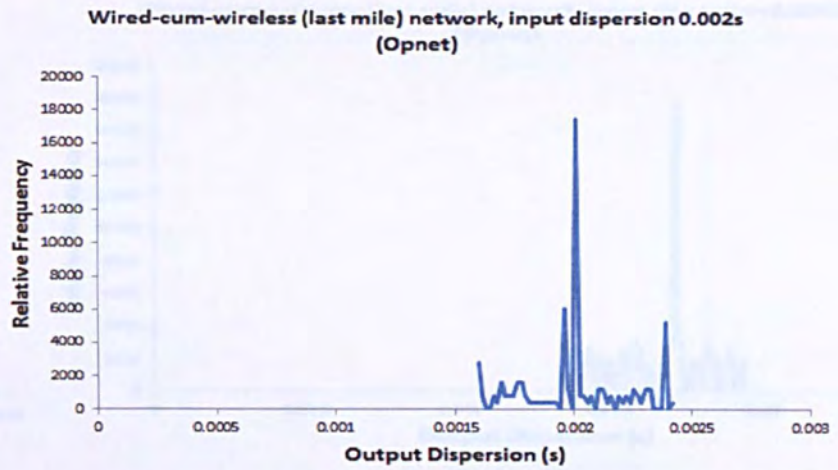
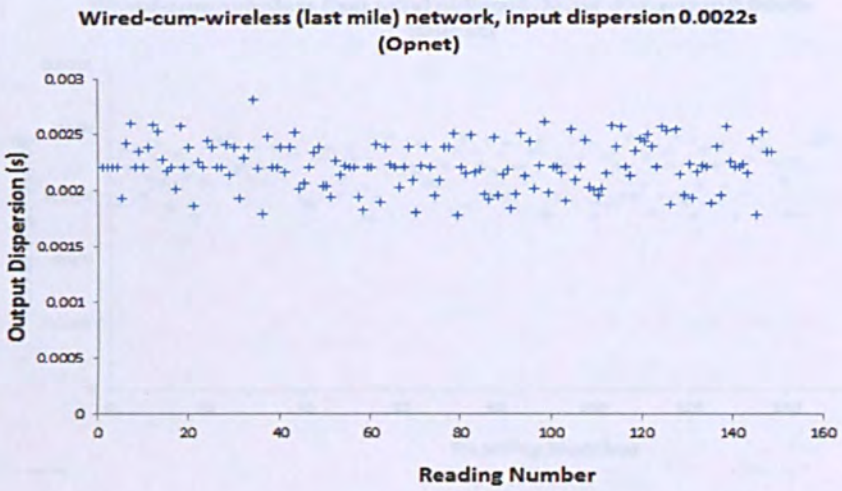


Figure 5.41 Raw dispersion data and histograms for wired-cum-wireless network using 256 byte probe packet pairs with input dispersions for 0.0014 and 0.0018 seconds.







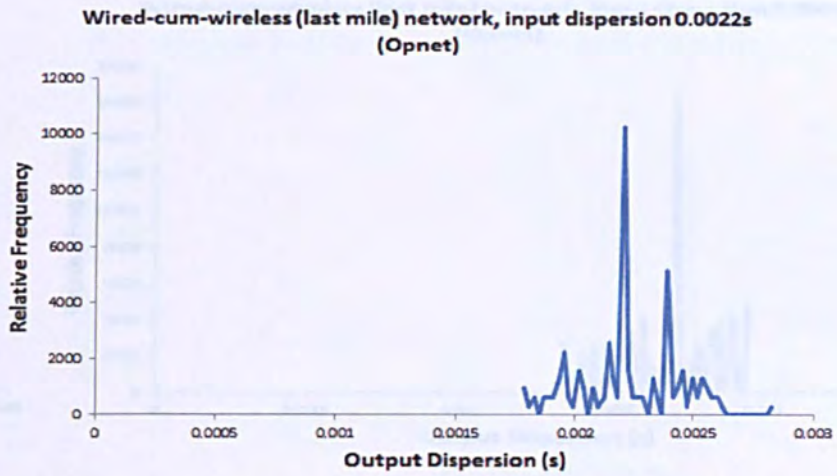


Figure 5.42 Raw dispersion data and histograms for wired-cum-wireless network using 256 byte probe packet pairs with input dispersions for 0.002 and 0.0022 seconds.

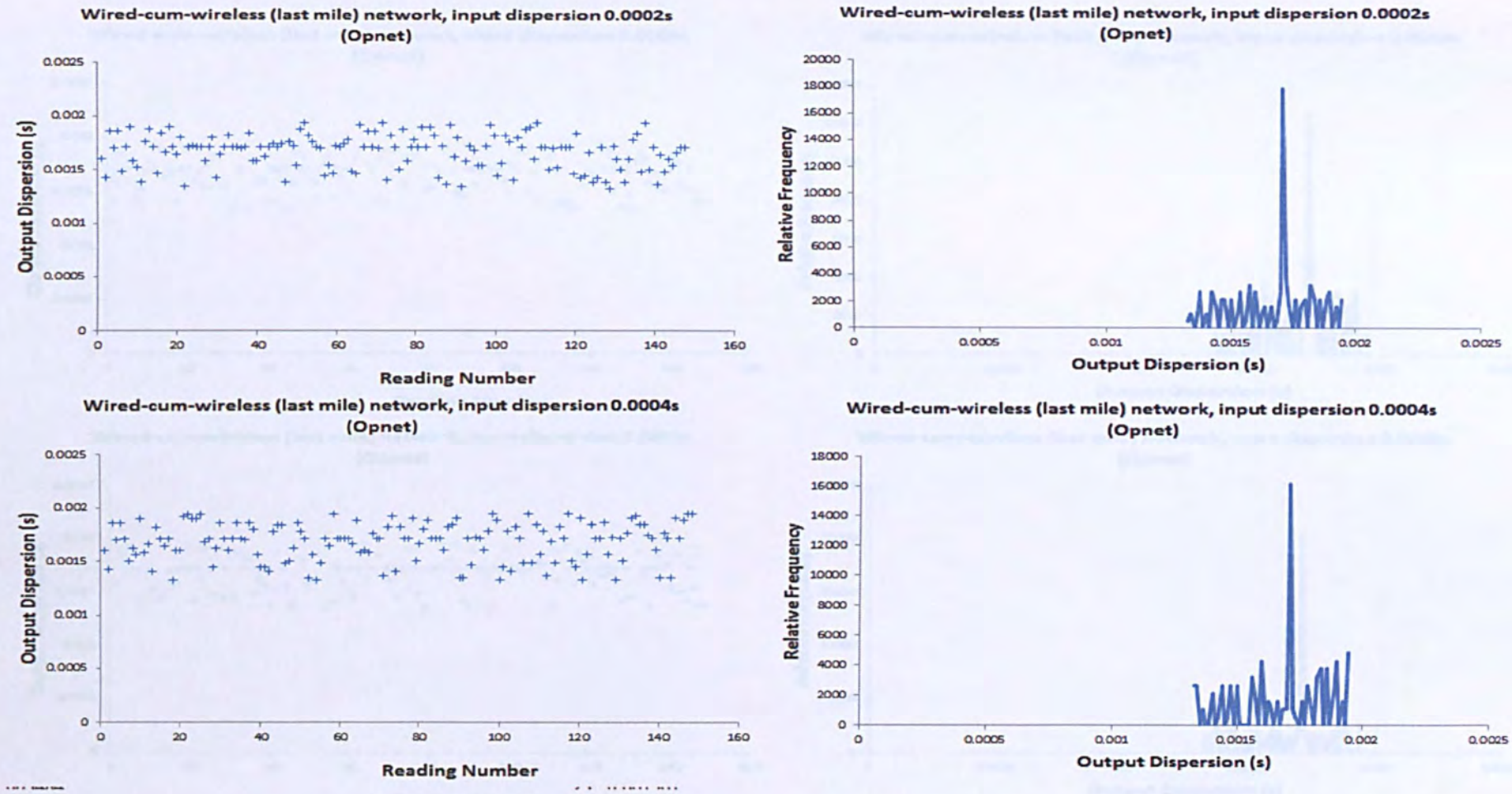


Figure 5.43 Raw dispersion data and histograms for wired-cum-wireless network using 1024 byte probe packet pairs with input dispersions for 0.0002 and 0.0004 seconds.

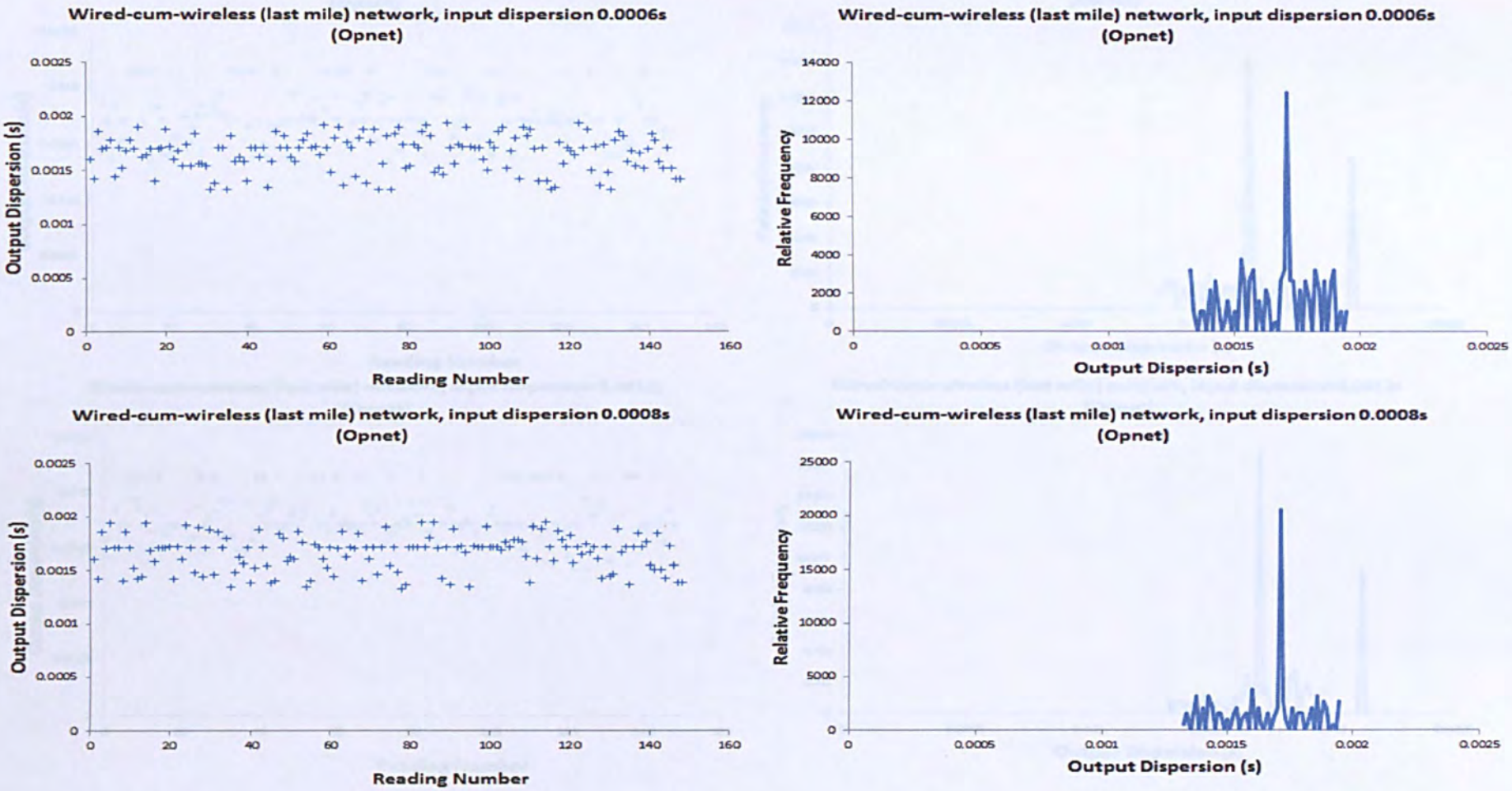


Figure 5.44 Raw dispersion data and histograms for wired-cum-wireless network using 1024 byte probe packet pairs with input dispersions for 0.0006 and 0.0008 seconds.

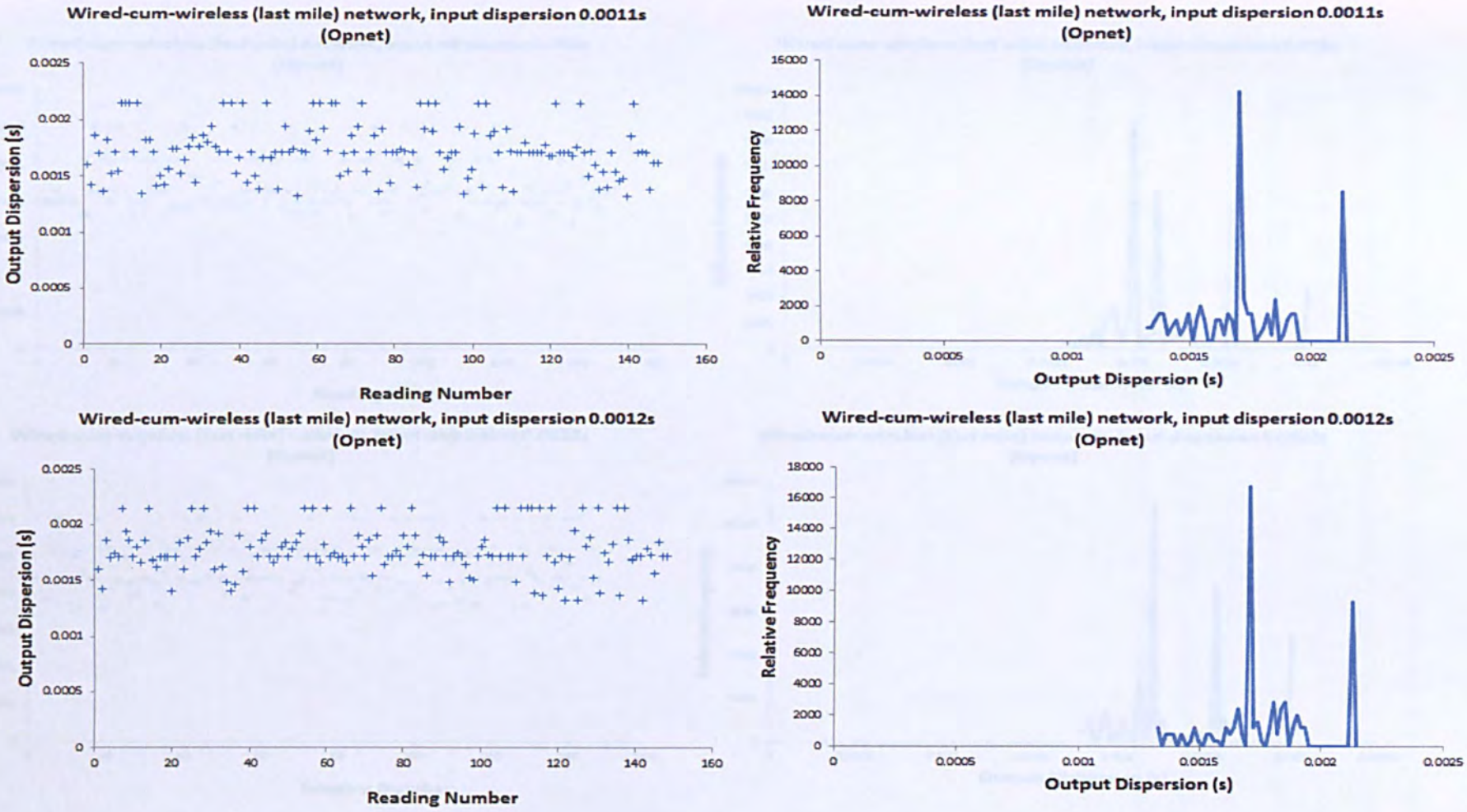


Figure 5.45 Raw dispersion data and histograms for wired-cum-wireless network using 1024 byte probe packet pairs with input dispersions for 0.0011 and 0.0012 seconds.

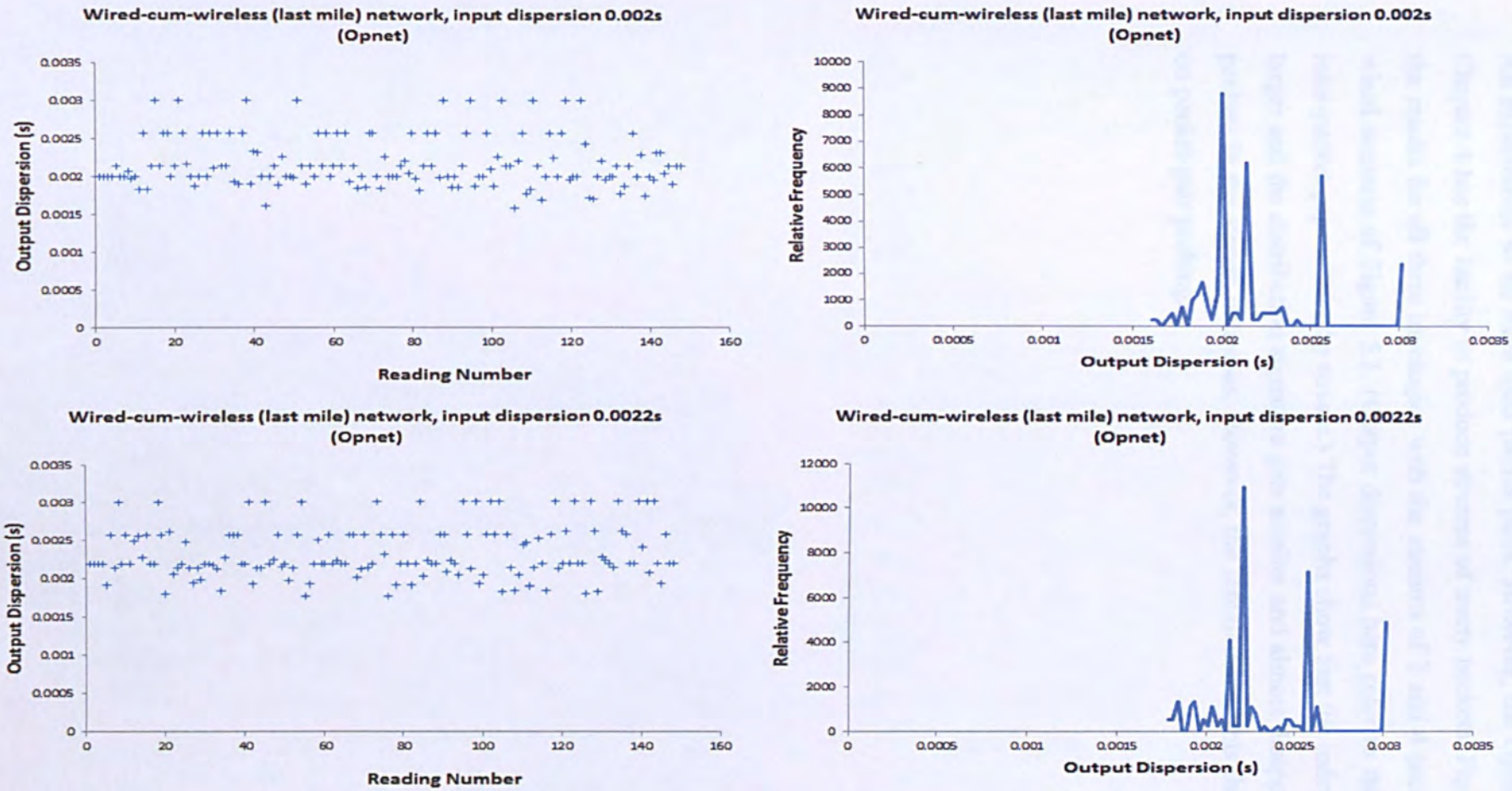


Figure 5.46 Raw dispersion data and histograms for wired-cum-wireless network using 1024 byte probe packet pairs with input dispersions for 0.002 and 0.0022 seconds.

5.3.5. Streams or more than two packets

All experiments so far have used packet pairs. However, the Opnet model developed in Chapter 4 has the facility to produce streams of many packets. Figure 5.47 to 5.49 show the results for all three topologies with the streams of 3 and 4 packets, using the simple wired scenario of Figure 5.1. (Output dispersions here refer to the mean spacing of all inter-packet gaps within the stream.) The graphs show that the independent signature gets larger and the distribution signature gets smaller and almost disappears as the number of packets in the stream increases. However, the remainder of this thesis concentrates only on packet-pair probing.

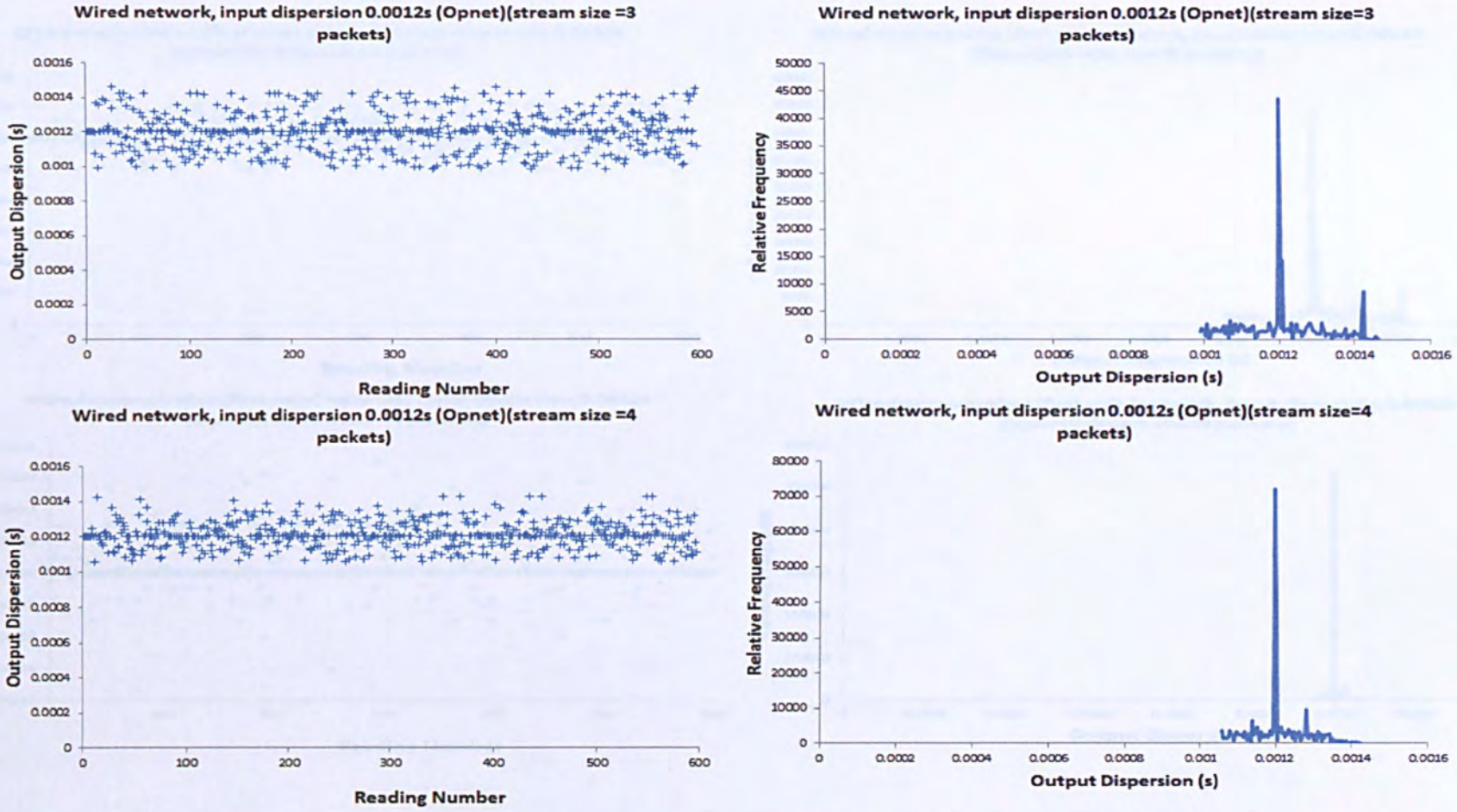


Figure 5.47 Raw dispersion data and histograms for wired network using 128 byte stream of 3 and 4 packets with input dispersions between 0.0012.

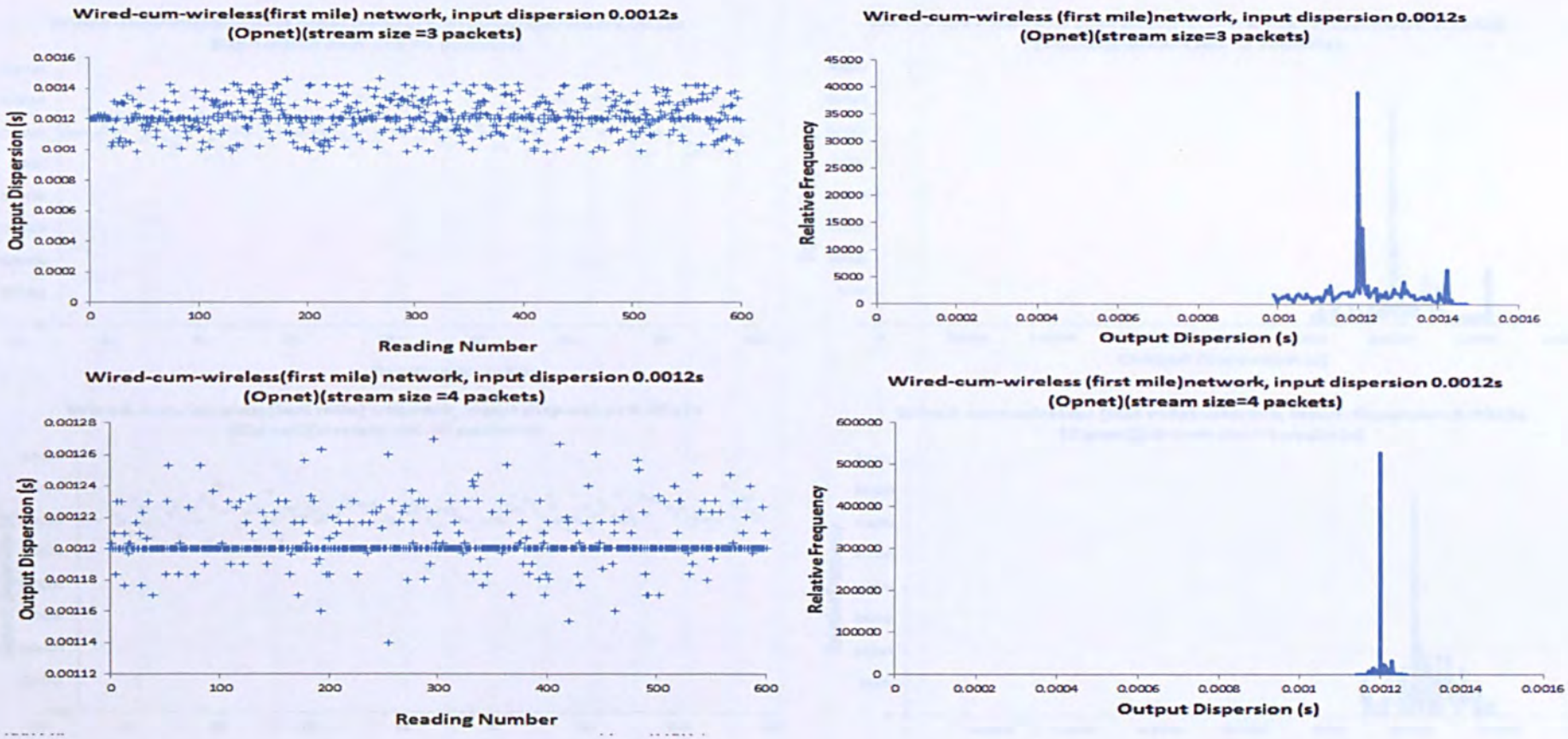


Figure 5.48 Raw dispersion data and histograms for wired-cum-wireless (first mile) network using 128 byte stream of 3 and 4 packets with input dispersions between 0.0012.

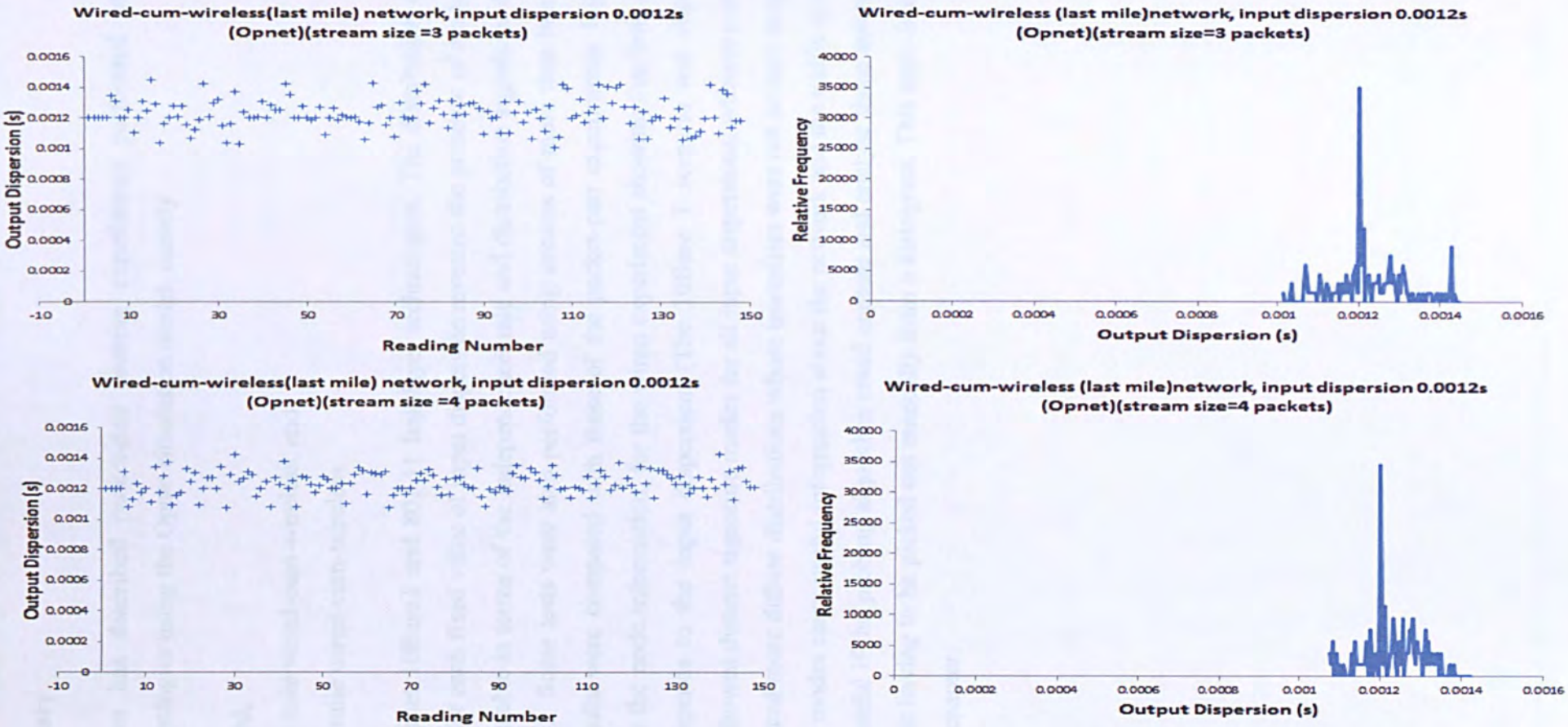


Figure 5.49 Raw dispersion data and histograms for wired wired-cum-wireless (last mile) network using 128 byte stream of 3 and 4 packets with input dispersions between 0.0012.

5.4. Summary

This chapter has described packet-pair probing experiments performed with three network topologies using the Opnet simulation model, namely

- wired,
- first mile wired-cum-wireless, and
- last mile wired-cum-wireless

Using Ethernet 10BaseT and 801.11 link-layer technologies. The distribution of output dispersion for each fixed value of input dispersion revealed the presence of modes, which were explainable in terms of the independence, rate and distribution signatures described in Chapter 3. Some tests were also performed using streams of more than two packets, and the results were compared with those of the packet-pair experiments. Figure 5.53 summarizes the node information, for the wired experiment showing that how signature intensity depends to the input dispersion. (The 10Base T scenario was chosen here because it showed distinct signature modes for all input dispersions: wireless experiments often produced more diffuse distributions whose intensities were not so easy to quantify.) Since these modes carry useful information about the network and the traffic it carries, it would be useful if the probing algorithm could capture and analyse these automatically, without their having to be picked out manually from a histogram. This topic is addressed in the next chapter.

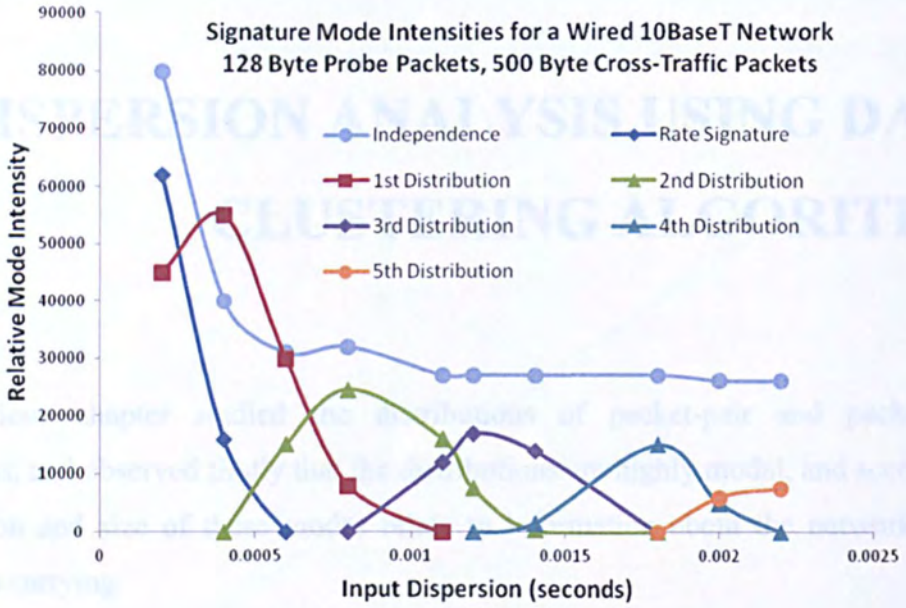


Figure 5.50 Relative mode intensities for wired experiment

6. DISPERSION ANALYSIS USING DATA-CLUSTERING ALGORITHMS

The previous chapter studied the distributions of packet-pair and packet-stream dispersions, and observed firstly that the distributions are highly modal, and secondly that the position and size of these modes relate to information about the network and the traffic it is carrying.

The technique used to study the distributions was the histogram method. However, as Lai and Baker (Lai and Baker 1999) have pointed out, there are some of the disadvantages to using the histogram method:

- Lack of foreknowledge: it is very difficult to choose a suitable bin width before the distribution itself has been studied.
- Possible relationships (i.e. membership of the same distribution mode) between data points on opposite sides of a bin boundary are ignored.
- Lack of relationship between data points on opposite sides of a bin is also ignored.

To overcome these problems, Chapter 5 used histograms with very small bin-size (1/100 of total measured dispersion range) to capture detail in the distribution. However this requires a large number of samples (in our case 600) to ensure a statistically significant number fall in to each bin. (Increasing the bin size would improve the statistical significance of each bin, but at the expense of blurring the fine features of the distribution.)

Figure 6.1 shows the time required to collect 600 packet-pair samples as a function of the time between pairs (in the Chapter 5 experiments this was set to 2 seconds). Figure 6.2 shows the bandwidth overhead (as a percentage of the 10Mbit/s link rate) as a function of inter-pair time for the three packet sizes used in Chapter 5. Thus we see that when the

time between samples is reduced in order to facilitate faster measurement, the bandwidth overhead increases. It is therefore desirable to estimate the distribution with fewer samples.

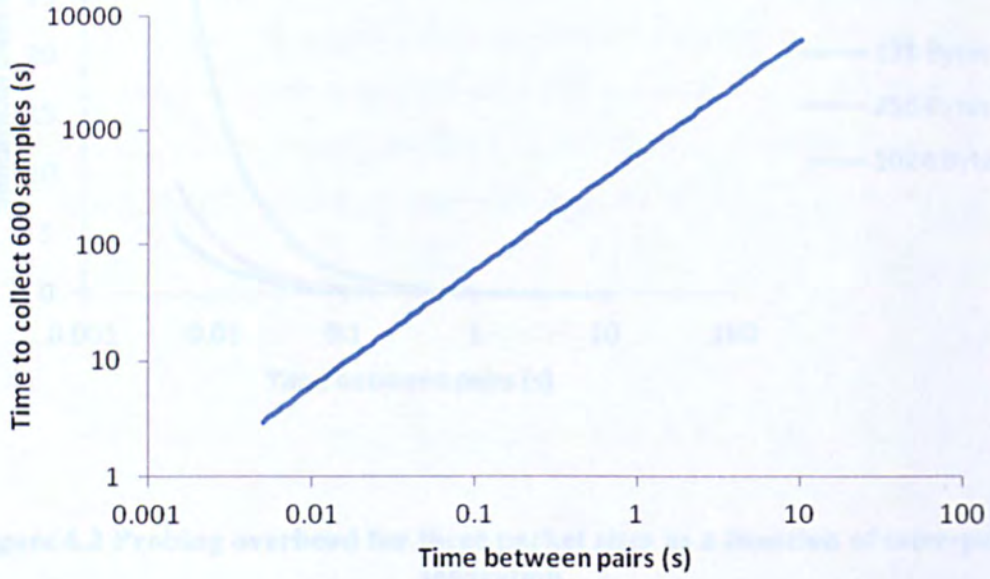


Figure 6.1 Time to collect 600 samples as a function of inter-packet-pair separation

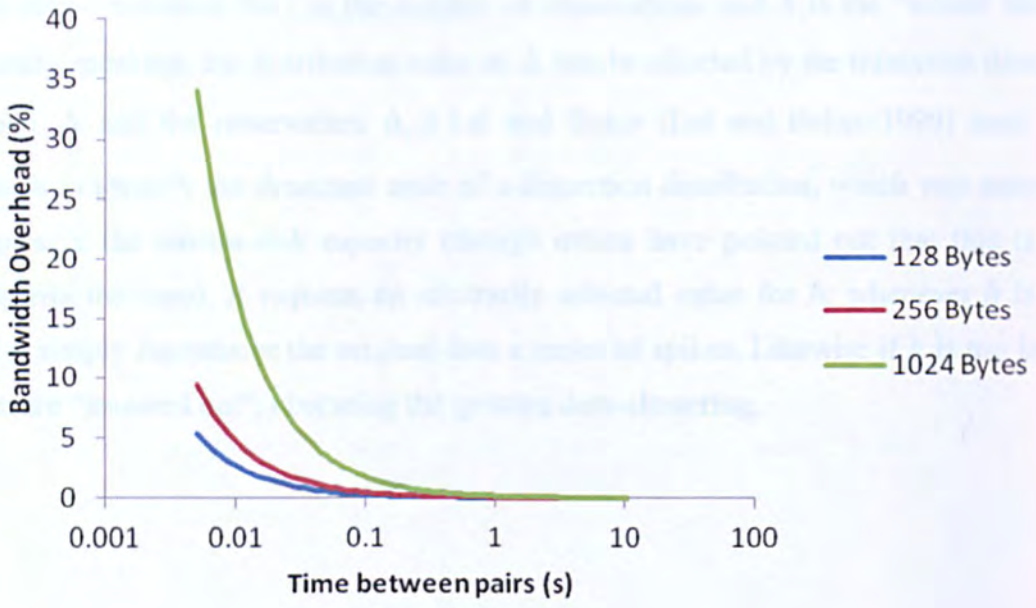


Figure 6.2 Probing overhead for three packet sizes as a function of inter-pair separation

6.1. Introduction

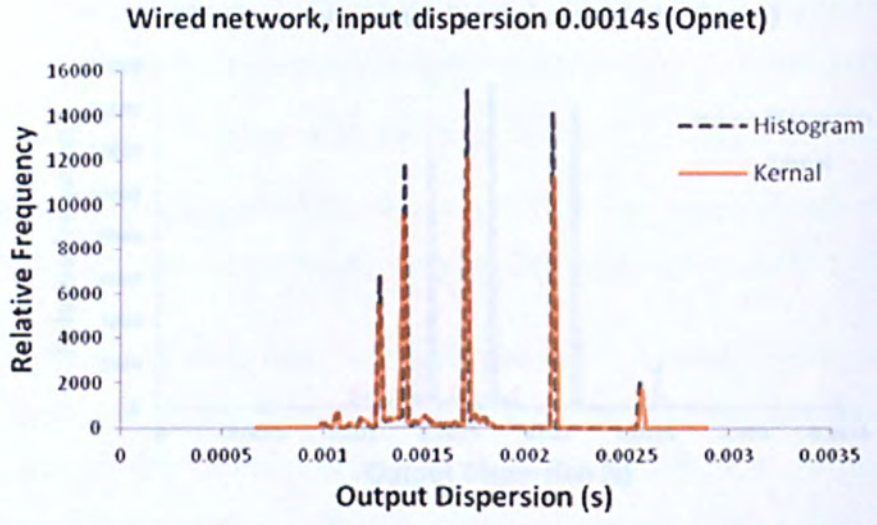
In this chapter we examine three alternatives to the histogram method, starting with the Kernel Density method of Lai and Baker (Lai and Baker 1999) . Throughout this chapter we use a common notation: the different samples of output dispersion Δ_{out} are denoted $\{\Delta_1, \Delta_2 ... \Delta_t\}$ where t is time expressed as the number of packet-pair transmissions since the experiment began.

6.2. The Kernel Density Estimator (KDE)

A well known technique for estimating the pdf of a random variable from a finite set of observations is Kernel-Density Estimation. This uses a "kernel function" $K(x)$ which is typically a Gaussian or symmetrical triangular function with zero mean, unity width/standard deviation and a unity integral between $\pm\infty$. The estimated distribution is

$$f(\Delta_t) \approx \frac{1}{t}\sum_{i=1}^{t}\frac{1}{h}K\left(\frac{\Delta_t - \Delta_i}{h}\right) \qquad (6.1$$

In the above equation the t is the number of observations and h is the "kernel width" (generally speaking, the distribution value at Δ can be affected by the minimum distance between Δ and the observation Δ_i.) Lai and Baker (Lai and Baker 1999) used this approach to identify the dominant node of a dispersion distribution, which was assumed to represent the narrow-link capacity (though others have pointed out that this is not necessarily the case). It requires an arbitrarily selected value for h: whenever h is too small it simply reproduces the original data a series of spikes. Likewise if h is too large, modes are "smeared out", obscuring the genuine data-clustering.



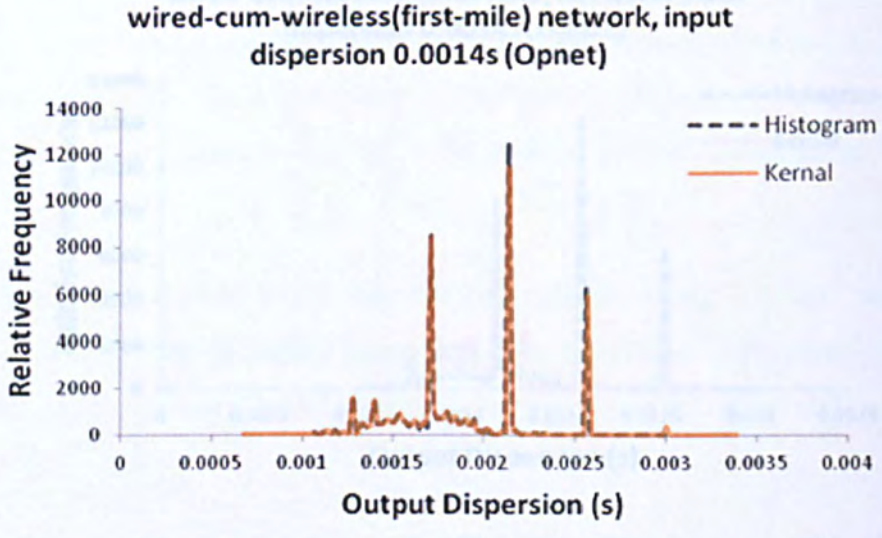


Figure 6.3 Kernel Density estimation dispersion distributions compared with corresponding histogram estimates from Chapter 5, based on 600 data points.

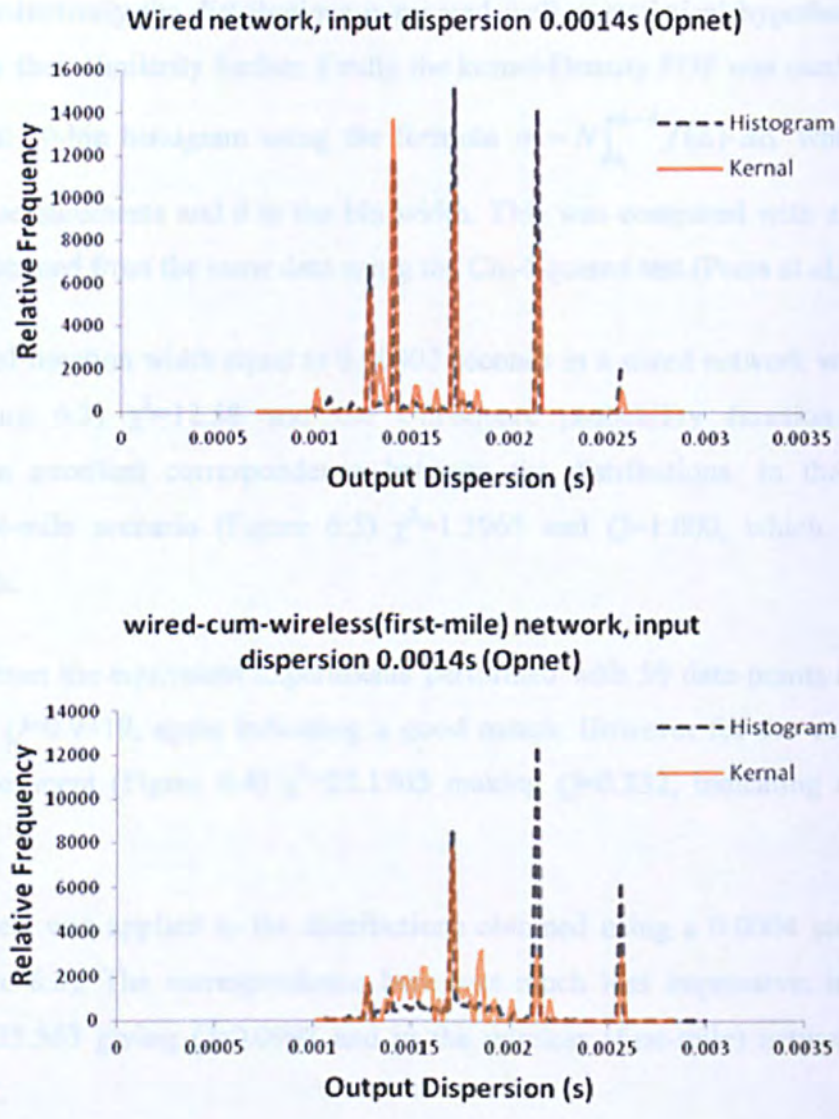


Figure 6.4 Kernel Density estimates for 50 points. (Histograms still based on 600 points).

Figures 6.3 and 6.4 show the compression between results for kernel density with 600 data points and 50 data points: with the 600 data points the result is almost identical to the histogram. When the kernel density estimate is based on only 50 data points, some of the modes are different in intensity, but are almost all still in the right places.

The kernel width for Figures 6.3 and 6.4 was set to 0.00002 seconds. Figure 6.5 shows the effect of increasing it to 0.00004 seconds; the modes are now noticeably wider and shorter, but still show a good compatibility with histogram.

Although qualitatively the distributions compared well, a statistical hypothesis test was used to study their similarity further. Firstly the kernel-Density PDF was used to generate an equivalent 20-bin histogram using the formula $n_i = N\int_{\Delta_i}^{\Delta_i+\delta} f(\Delta)\cdot d\Delta$ where N is the number of measurements and δ is the bin width. This was compared with a real 20-bin histogram obtained from the same data using the Chi-Squared test (Press et al, 1992).

For the kernel function width equal to 0.00002 seconds in a wired network with 600 data points (Figure 6.3) χ^2=12.58 and the Chi-square probability function Q=0.9319. indicating an excellent correspondence between the distributions. In the equivalent wireless first-mile scenario (Figure 6.3) χ^2=1.3965 and Q=1.000, which is almost a perfect match.

Using data from the equivalent experiments performed with 50 data points (Figure 6.4) χ^2=9.72 and Q=0.9319, again indicating a good match. However for the corresponding wireless experiment (Figure 6.4) χ^2=22.1705 making Q=0.232, indicating a less exact match.

Finally the test was applied to the distributions obtained using a 0.0004 second kernel width (Figure 6.5). The correspondence here was much less impressive: in the wired network χ^2=25.563 giving Q=0.0998 and in the wireless (first-mile) network χ^2=24.63 and Q=0.155.

Although the kernel density method predicts the PDF of the dispersion, it does not make the mode sizes and position directly accessible as entities within the software. We therefore investigate a second method, namely the E-M k-Means algorithm.

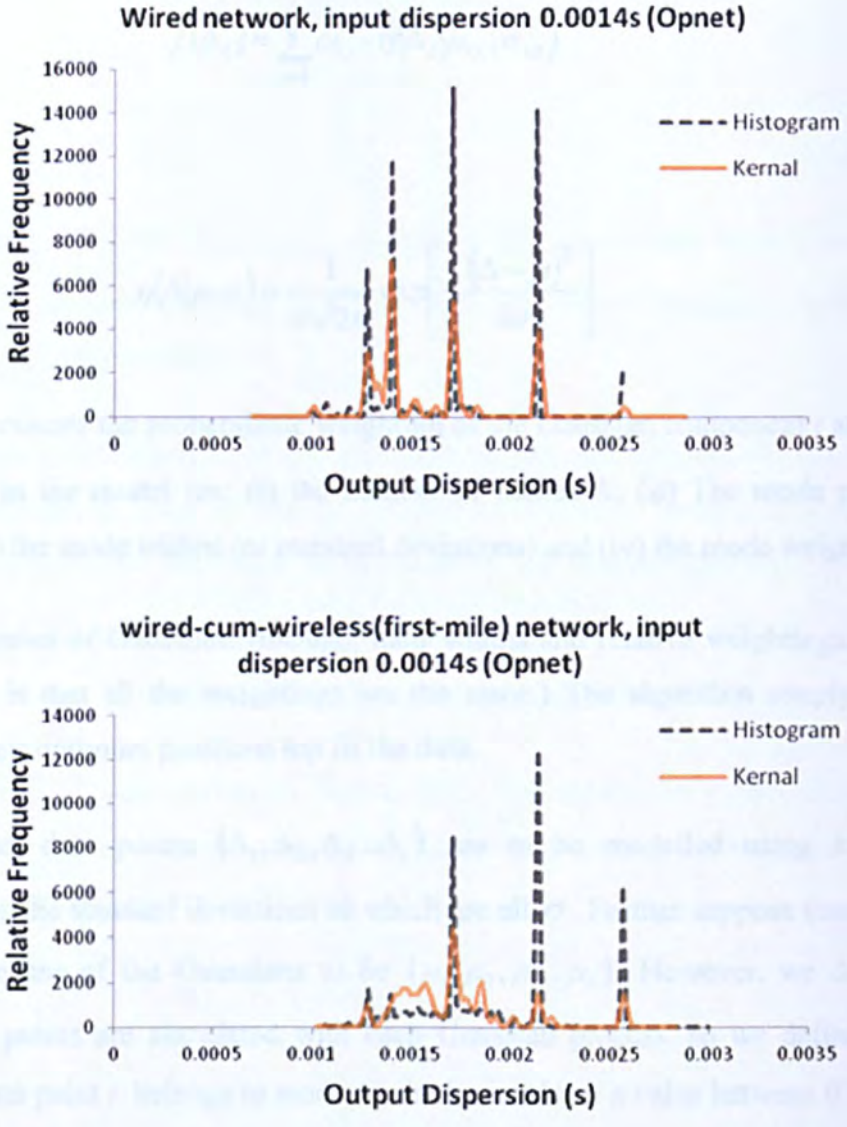


Figure 6.5 Kernel Density estimates for 50 points, kernel width 0.00004.

6.3. The Gaussian E-M k-Means Algorithm

This is a different approach to data clustering than the Kernel Density method. It was originally devised by Dempster et al.(Dempster, Laird et al. 1977), but the analysis here is based on Mitchell (Mitchell 1997). Here we represent the pdf for Δ_t as a weighted sum of k Gaussian distributions:

$$f(\Delta_t) = \sum_{i=1}^{k} \omega_{i,t} \cdot \eta\left(\Delta_t \middle| \mu_{i,t}, \sigma_{i,t}\right) \tag{6.2}$$

where

$$\eta(\Delta|\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{(\Delta-\mu)^2}{2\sigma^2}\right] \tag{6.3}$$

and $\omega_{i,t}$ represents the probabilistic weighting of the Gaussian component i at time t. The parameters in the model are: (i) the number of modes K, (ii) The mode positions (or means), (iii) the mode widths (or standard deviations) and (iv) the mode weightings.

Assume number of Gaussians (means), their widths and relative weightings. (The usual assumption is that all the weightings are the same.) The algorithm simply moves the means to their optimum positions top fit the data.

Suppose that data points $(\Delta_1, \Delta_2, \Delta_3 .. \Delta_t)$ are to be modelled using k Gaussian distributions, the standard deviations of which are all σ. Further suppose that we initially guess the means of the Gaussians to be $(\mu_1, \mu_2, \mu_3 ... \mu_k)$. However, we do not know which data points are associated with each Gaussian process, so we define a "belief" value B_{ij} data point i belongs to mode j (which must have a value between 0 and 1.) This can be computed using the formula

$$B_{ij} = \frac{e^{-\frac{1}{2\sigma^2}(\Delta_i - \mu_j)^2}}{\sum_{n=1}^{k} e^{-\frac{1}{2\sigma^2}(\Delta_i - \mu_n)^2}} \tag{6.4}$$

(a corollary of Bayes' Theorem with a uniform prior.) Now with the belief values computed, the mean positions $(\mu_1, \mu_2, \mu_3 ... \mu_k)$ can be adjusted using the formula:

$$\mu_j = \frac{\sum_{i=1}^{m} B_{ij} \cdot \Delta_i}{\sum_{i=1}^{m} B_{ij}} \quad . \tag{6.5}$$

With the new values for $(\mu_1, \mu_2, \mu_3 ... \mu_k)$, the belief values may be re-computed using Eqn.(6.4). The process continues in a loop, and after 5 or 6 iterations, the modes settle into their most likely positions.

However, the initial position of the Gaussian means and standard deviations must be arbitrarily chosen. In order to seed the Gaussians uniformly throughout the data range, their initial means were calculated from the mean and standard deviation (μ, σ) of the data set. Means were arranged uniformly around μ, spaced σ seconds apart, and the standard deviations of the Gaussian components were all set to $\sigma/20$.

Some typical results are shown in Figures 6.6. The modes are mostly in their correct positions, though sometimes the algorithm became confused, a situation becomes worth when fewer data points are used (see Figure 6.7).

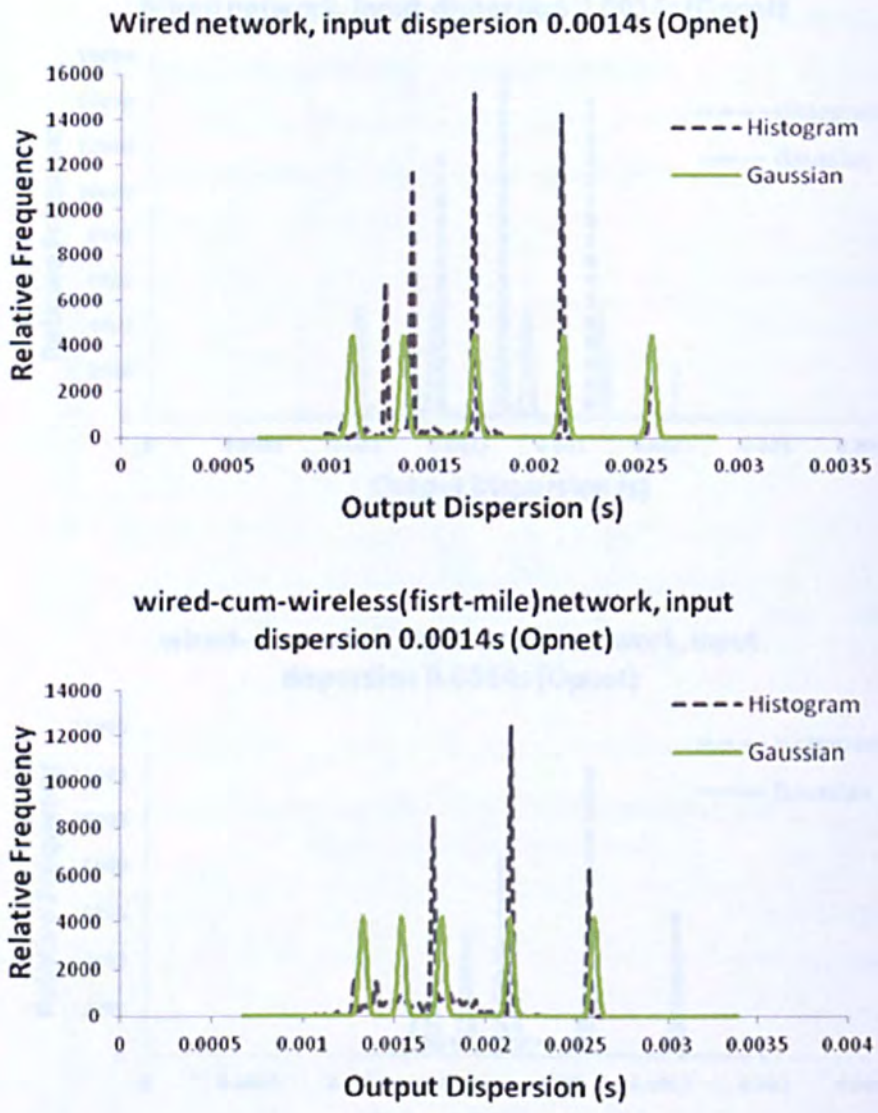


Figure 6.6 Dispersion distributions estimated using the E-M k-Means algorithms with 5 Gaussian components, compared with corresponding histograms. The Gaussian mode positions are based on all 600 data points.

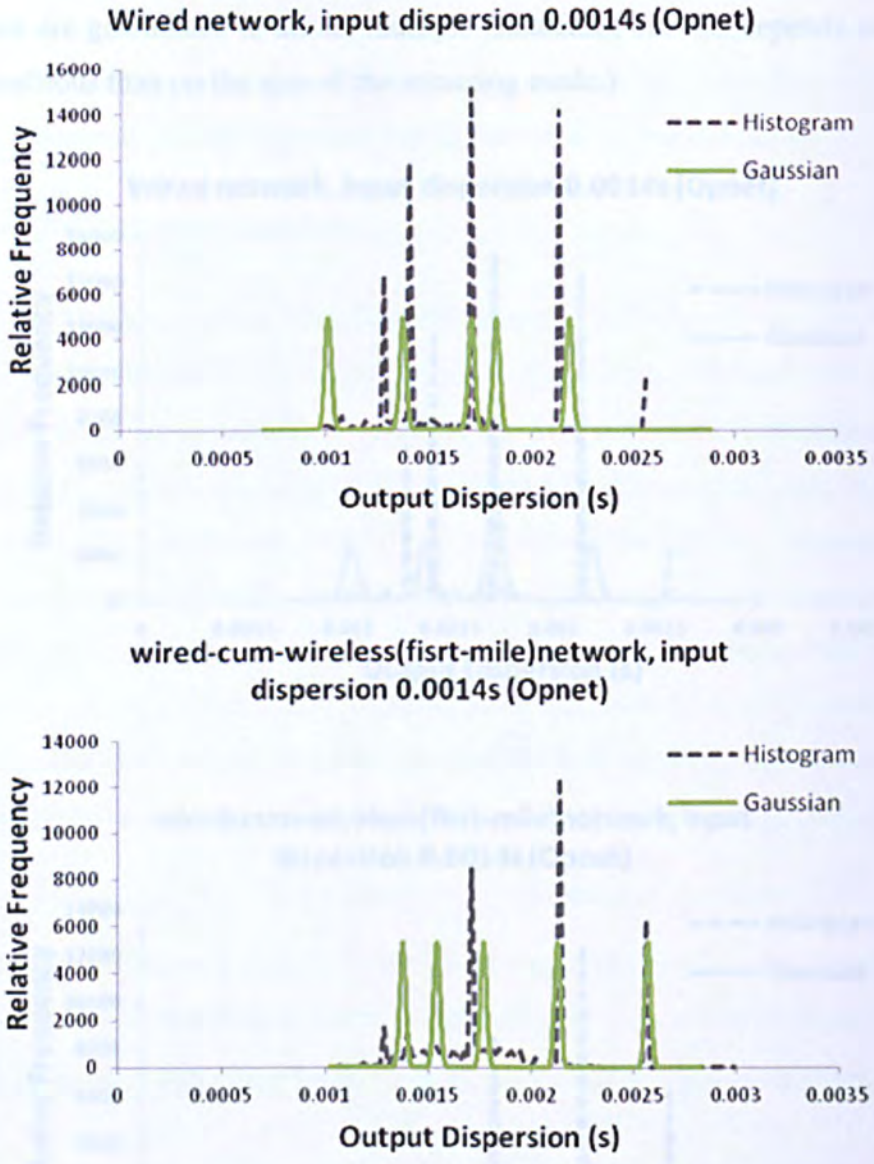


Figure 6.7 Dispersion distributions estimated using the E-M k-Means algorithms with 5 Gaussian components, compared with corresponding histograms. The Gaussian mode positions are based on only 50 data points.

In some further experiments the Gaussian standard deviation was increased to one tenth that of the data set. Figure 6.8 shows the results, modes are still almost in correct position but obviously twice as wide as before. Whereas all the modes in the previous experiment were of the same size in height, on wired network graph one mode is double the height of the others. This is because two Gaussian components have become attracted to the same data mode and have joint together to form a higher mode. (This effect is beneficial if

strong modes are guaranteed to attract multiple Gaussians, but this depends as much on their seed positions than on the size of the attracting mode.)

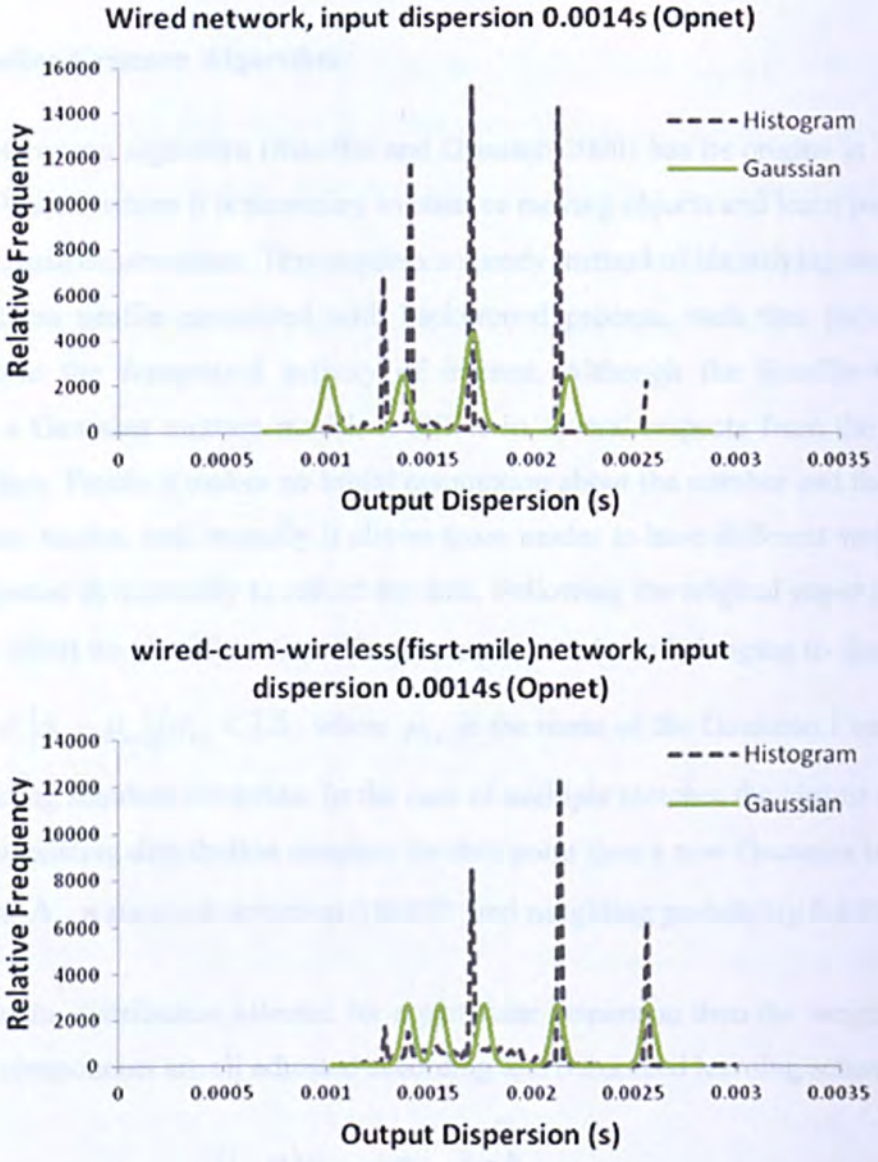


Figure 6.8 E-M k-Means experiment repeated with a larger Gaussian standard deviation.

On the whole the E-M k-Means Algorithm proves to be quiet fast and efficient; on average the algorithm takes only 8-9 iterations for the Gaussian means to converge to within 1% of their final values. Also the modes are available as entities (or objects) within the software and may easily be mapped to known probing signatures. On the other hand, although the modes often gravitate to their correct positions, the estimated PDF is generally a poor representation of the true distribution: unless multiple components are

lucky enough to converge on a strong mode (as in Figure 6.8, top), all the represented modes have the same sizes irrespective of their true intensities. Also the mode standard deviation is fixed, and may not represent the widths of the actual data modes.

6.4. The Stauffer-Grimson Algorithm

The Stauffer-Grimson algorithm (Stauffer and Grimson 2000) has its origins in the field of Computer Vision, where it is necessary to observe moving objects and learn patterns of activity from those observations. This requires a speedy method of identifying modes in a pixel's brightness profile associated with background process, such that they can be eliminated from the foreground activity of interest. Although the Stauffer-Grimson method uses a Gaussian mixture model, it differs in several respects from the E-M k-Means algorithm. Firstly it makes no initial assumption about the number and the widths of the Gaussian modes, and secondly it allows those modes to have different weightings which are adjusted dynamically to reflect the data. Following the original paper (Stauffer and Grimson 2000) we classify a dispersion measurement Δ_t as belonging to distribution i if and only if $|\Delta_t - \mu_{i,t}|/\sigma_{i,t} < 2.5$, where $\mu_{i,t}$ is the mean of the Gaussian i and $\sigma_{i,t}$ is the corresponding standard deviation. In the case of multiple matches the closest match is selected. If *no* existing distribution matches the data point then a new Gaussian is created with a mean of Δ_t, a standard deviation 0.00002 and weighting probability 0.0001.

If k represents the distribution selected for a particular dispersion then the weightings of the Gaussian components are all adjusted according to a reinforced learning scheme:

$$\omega_{i,t} = \begin{cases} (1-\alpha)\omega_{i,t-1} + \alpha; & i = k \\ (1-\alpha)\omega_{i,t-1}; & i \neq k \end{cases}$$

where α is the learning rate (which we initially set to 0.01) and renormalize such that the weightings again sum to unity. Adjustments to $\mu_{i,t}$ and $\sigma_{i,t}$ are applied only to the matched distribution, i.e.

$$\mu_{k,t} = (1-\rho)\mu_{k,t-1} + \rho\Delta_t$$

and

$$\sigma_{k,t} = \sqrt{(1-\rho)\sigma_{k,t-1}^2 + \rho(\Delta_t - \mu_{k,t})^2} \qquad (6.$$

where ρ is a second learning rate, which is adjusted according to the degree to which the new measurement fits the distribution. For this purpose Stauffer and Grimson (Stauffer and Grimson 2000) used the rule $\rho = \alpha \cdot \eta(\Delta_t | \mu_{k,t}, \sigma_{k,t})$, but this creates problems for very narrow distributions which require $\rho > 1$. Here we use a "compromise" formula which makes ρ decrease as the quality of the fit deteriorates, but ensures that it never exceeds α :

$$\rho = \alpha \cdot \exp\left[-\frac{(\Delta-\mu)^2}{2\sigma^2}\right]. \qquad (6.$$

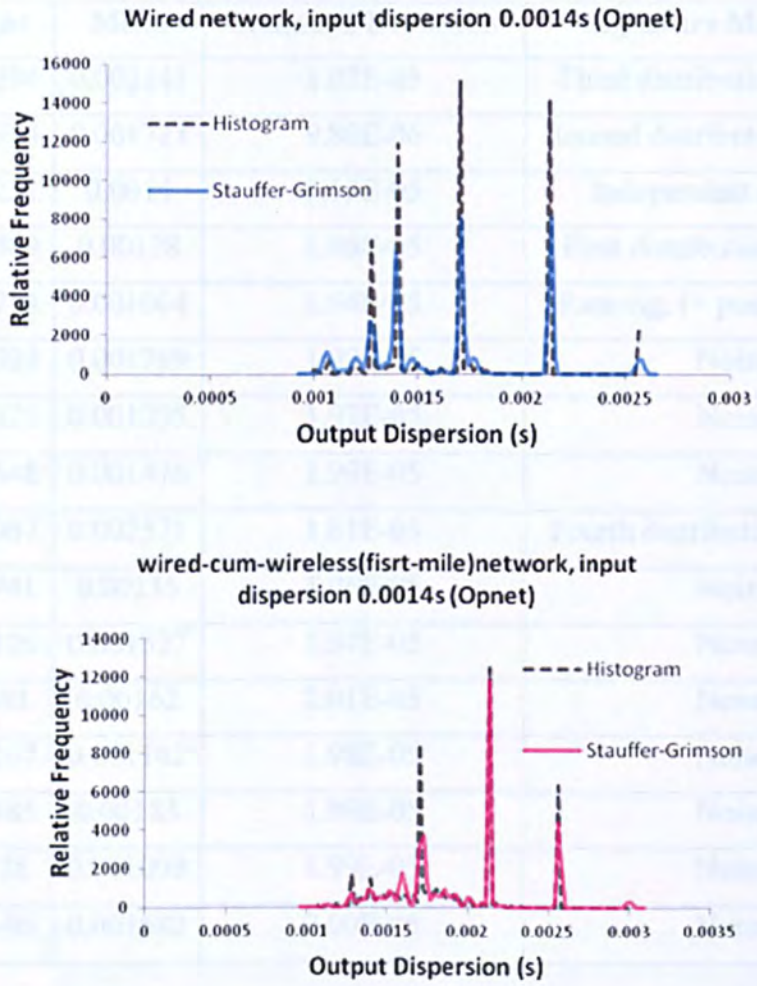


Figure 6.9 Stauffer-Grimson estimation of dispersion distributions based on 600 data points, compared with corresponding histograms from Chapter 5. Learning rate medium (α =0.01).

Some typical results are shown in Figure 6.9. The modes are almost in their correct positions, unlike those of the E-M k-Means algorithm (which was somewhat confused by the wireless network data).

Table 6.1 and 6.2 show the mode parameters (mean and standard deviation) listed in decreasing weight-order, mapped to the associated network signatures for wired and wired-cum-wireless network (see Table 5.3). It is clear that many of the smaller modes exist in response to noise (either independence or random back-off) rather than to specific signatures.

Weight	Mean	Standard Deviation	Signature Mechanism
0.224294	0.002141	1.02E-05	Third distribution signature
0.197974	0.001711	9.80E-06	Second distribution signature
0.189221	0.0014	1.15E-05	Independent signature
0.107409	0.00128	1.46E-05	First distribution signature
0.053714	0.001064	1.94E-05	Rate sig. (+ possible noise)
0.044303	0.001769	1.99E-05	Noise
0.040828	0.001205	1.97E-05	Noise
0.038648	0.001476	1.99E-05	Noise
0.038067	0.002571	1.81E-05	Fourth distribution signature
0.028741	0.00135	1.96E-05	Noise
0.011126	0.001527	1.97E-05	Noise
0.01083	0.00162	2.01E-05	Noise
0.009107	0.001142	1.98E-05	Noise
0.003885	0.00183	1.99E-05	Noise
0.00178	0.001009	1.99E-05	Noise
7.32E-05	0.001682	2.00E-05	Noise

Table 6.1 Table of mode parameters for wired network, produced by Stauffer-Grimson algorithm using 0.01 learning rate.

Weight	Mean	Standard Deviation	Signature Mechanism
0.282936	0.002141	9.37E-06	Third distribution signature
0.269205	0.00173	2.14E-05	Second distribution signature
0.083915	0.002571	1.45E-05	Fourth distribution
0.055282	0.0014	1.83E-05	Independent signature
0.054522	0.001523	1.93E-05	Noise
0.051368	0.001462	1.97E-05	Noise
0.047201	0.001801	1.86E-05	Noise

0.043097	0.00128	1.86E-05	First distribution
0.040189	0.001861	1.93E-05	Noise
0.022591	0.001602	1.98E-05	Noise
0.019902	0.001921	1.95E-05	Noise
0.009762	0.00168	1.96E-05	Noise
0.0095	0.002012	1.97E-05	Noise
0.005192	0.001346	1.94E-05	Noise
0.001919	0.002205	2.00E-05	Noise
0.00191	0.001131	2.01E-05	Noise
0.001228	0.003002	1.99E-05	Noise
0.000152	0.00228	1.99E-05	Noise
0.000118	0.001229	1.99E-05	Noise
5.88E-06	0.00234	2.00E-05	Noise
4.43E-06	0.002096	2.00E-05	Noise
1.19E-06	0.00106	2.00E-05	Noise

Table 6.2 Table of mode parameters for wired-cum wireless network produced by Stauffer-Grimson algorithm using 0.01 learning rate.

Figure 6.10 is shows the same experiment with fewer data (50 data points). In wired network (top) a lot of early activity on the independent signature (0.0014s) has given this node an early strength (this can be seen in Figure 5.9) not representative of its overall performance across the experiment. As a result of this the other modes have been substantially reduced by the normalisation process. Similarly in the wired-cum-wireless network (bottom) there has been much activity on the third distribution signature (0.002141s) between readings 150 and 200 (see Figure5.21) and the weight of this node has become very large. The rest of the modes have subsequently lost weight.

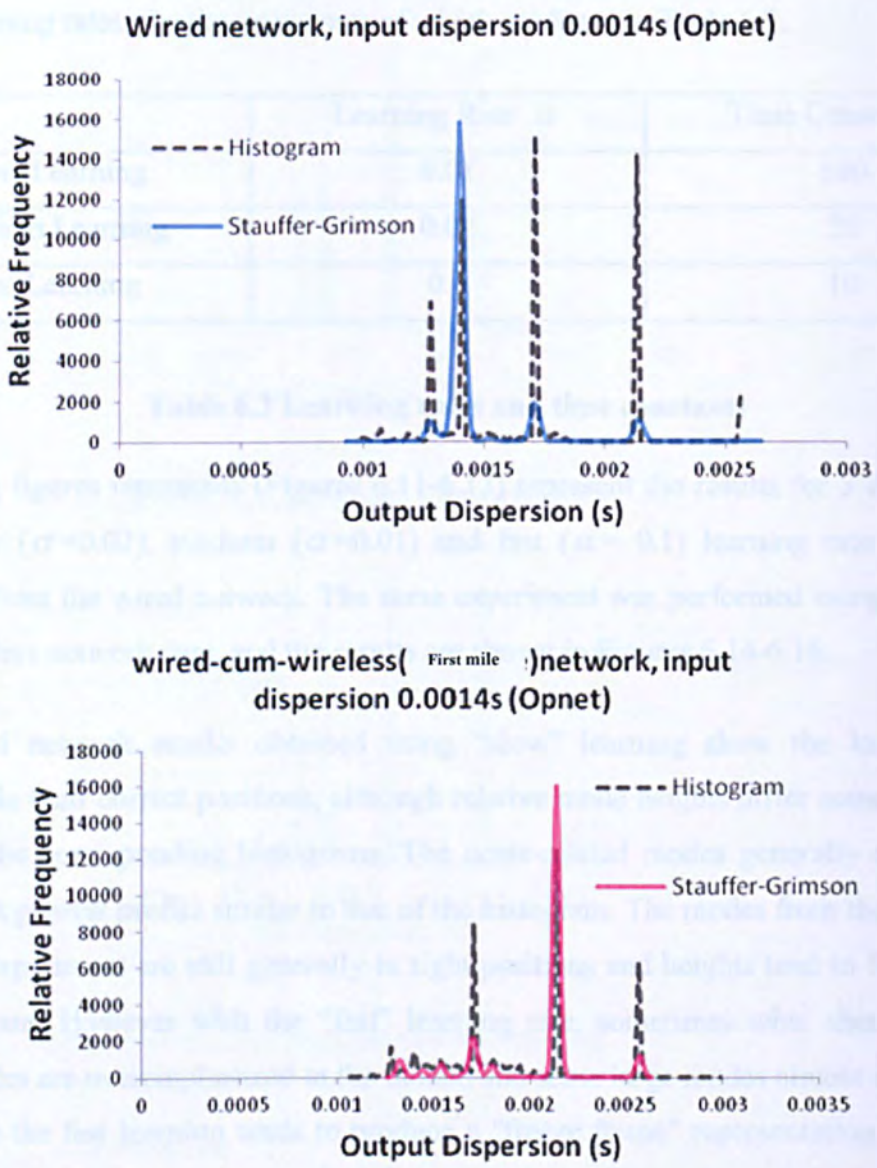


Figure 6.10 Stauffer-Grimson estimation of dispersion distributions based on 50 data points, compared with corresponding histograms from Chapter 5. Learning rate medium (α =0.01).

Now in this model, the mode heights are effectively governed by a first-order low pass filter, whose time constant depends on the learning rate. It is easy to show that when α is small, the filter time constant $T \approx 1/\alpha$ sampling periods. The time constants for α=0.01 is therefore 100 samples, so a modes can persist even if unsupported by incoming data for

a good portion of the 600-sample experiment time. We now experiment with higher and lower learning rates, the time constants of which are listed in Table 6.3.

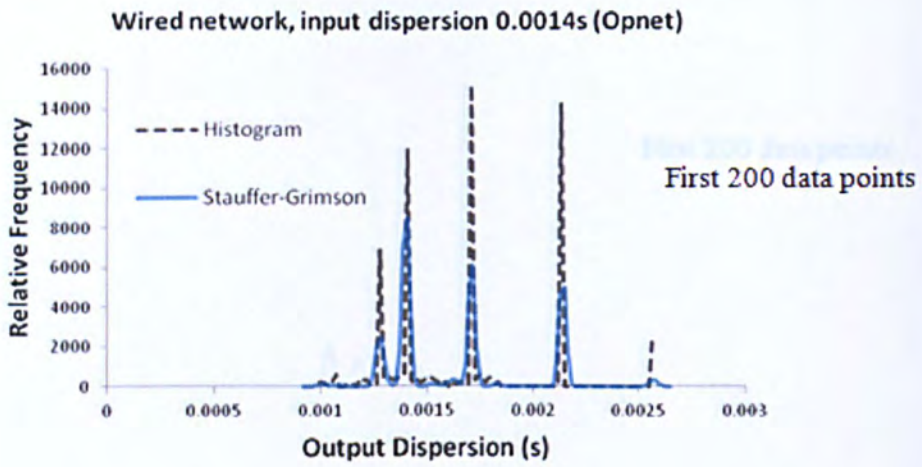
	Learning Rate α	Time Constant T
Slow Learning	0.01	100
Medium Learning	0.02	50
Fast Learning	0.1	10

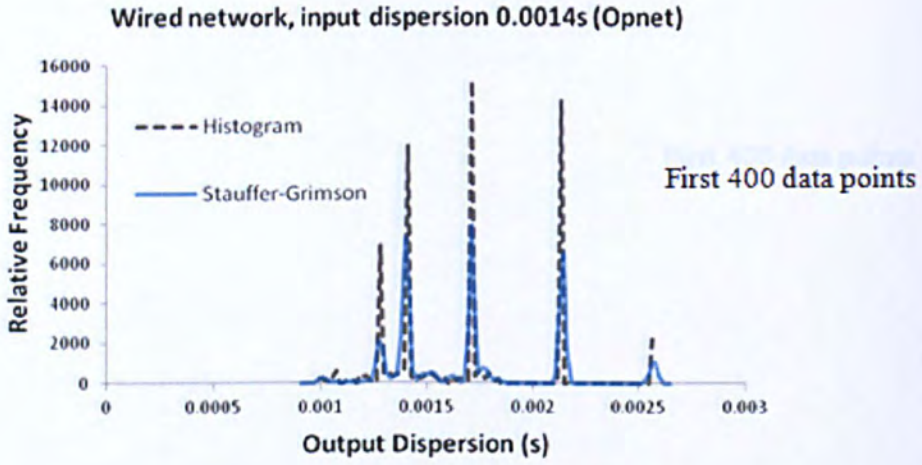
Table 6.3 Learning rates and time constants

Following figures represents (Figures 6.11-6.13) represent the results for 3 distributions with slow (α=0.02), medium (α=0.01) and fast (α= 0.1) learning rate using data obtained from the wired network. The same experiment was performed using the wired-cum-wireless network data, and the results are shown in Figures 6.14-6.16.

The wired network results obtained using "slow" learning show the larger modes generally in their correct positions, although relative mode heights differ sometimes from those of the corresponding histograms. The noise-related modes generally quite small, and form a general profile similar to that of the histogram. The modes from the "medium" learning experiment are still generally in right positions and heights tend to follow those of histogram. However with the "fast" learning rate, sometimes what should be very small modes are overemphasized in the model, and some large modes almost vanish. This is because the fast learning tends to produce a "freeze frame" representation of activity, rather than a long term average: modes which have experienced recent high activity are heavily emphasized in the model, even though their average activity across the experiment may be relatively small.

The results for the wired-cum–wireless network are generally similar to those of the wired network: slow and medium learning both give quite accurate models, while the fast learning produces great discrepancies. However, the slow learning result after 200 readings produces a curiously large and quite wide mode at the independence signature position. This shows that even with slow learning, ephemeral activity can cause the algorithm to produce unrepresentative results.





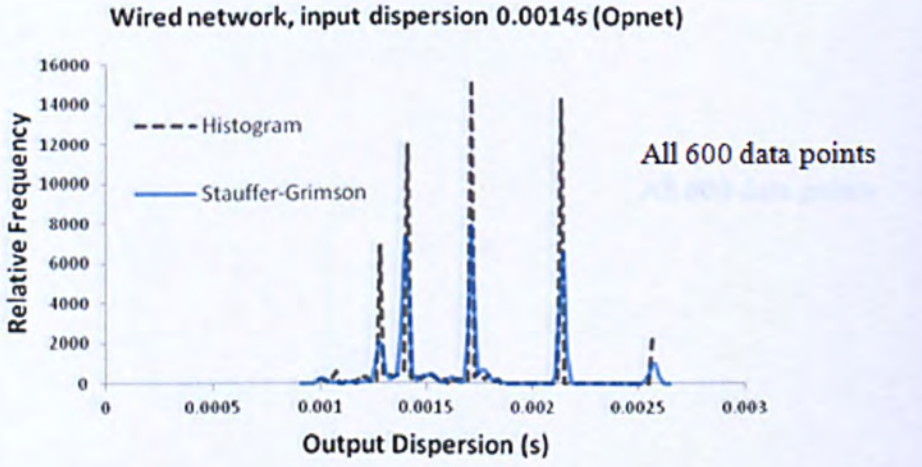


Figure 6.11 Stauffer-Grimson estimation ("slow" learning rate, α =0.01) of the dispersion distributions from the wired network compared with corresponding histogram estimates.

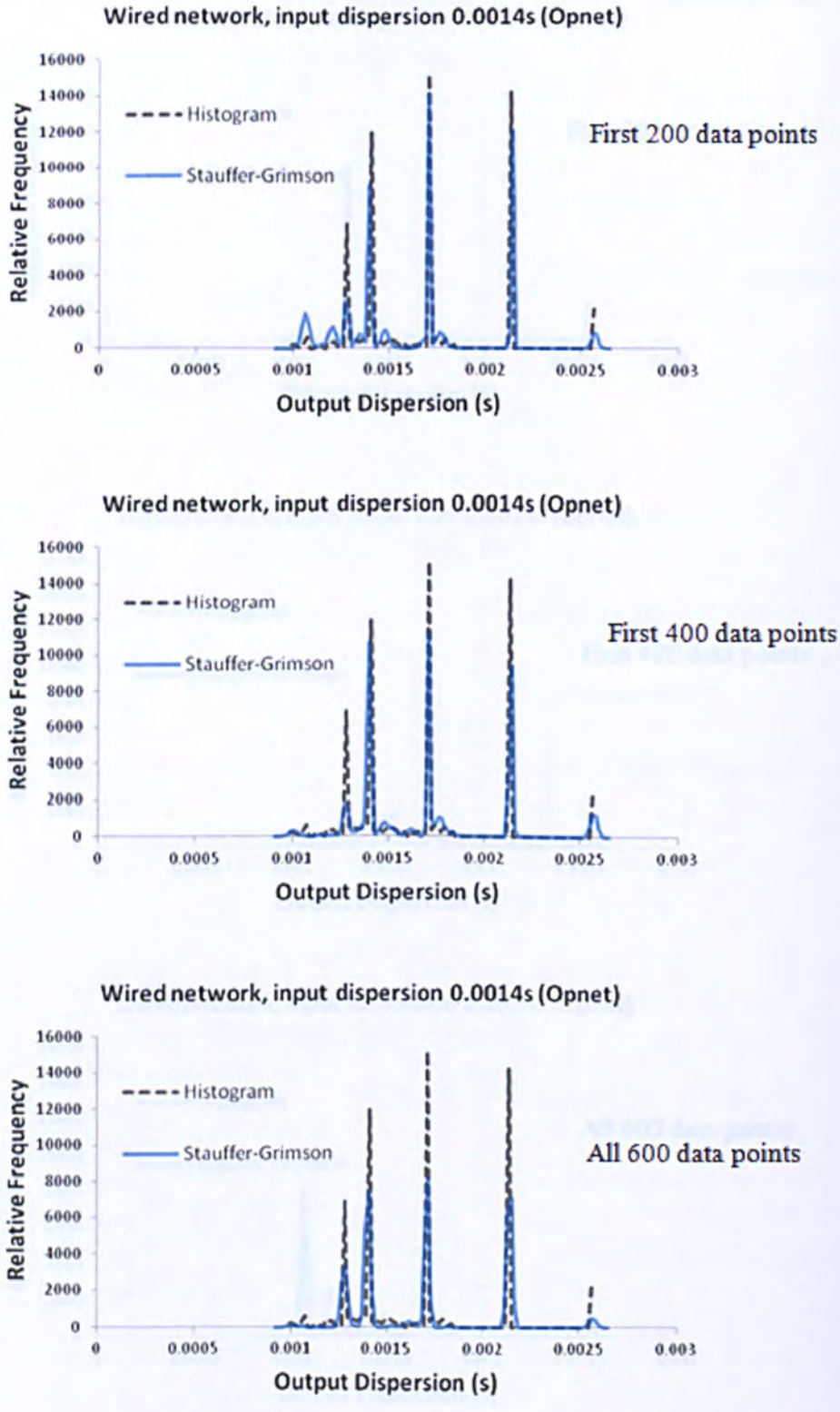


Figure 6.12 Stauffer-Grimson estimation ("medium" learning, α =0.02) of the dispersion distributions from the wired network compared with corresponding histogram estimates.

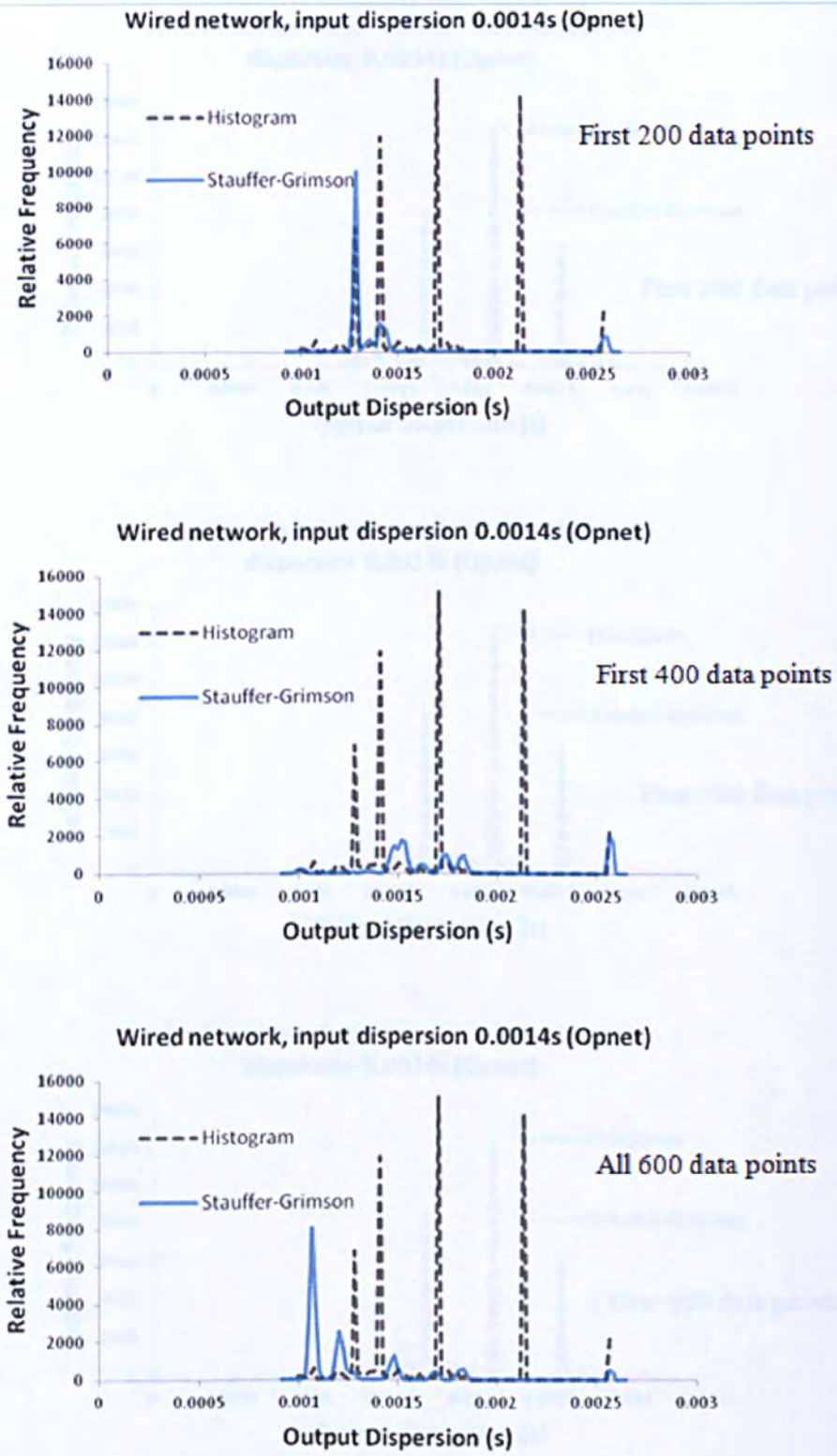


Figure 6.13 Stauffer-Grimson estimation ("fast" learning rate, α =0.1) of the dispersion distributions from the wired network compared with corresponding histogram estimates

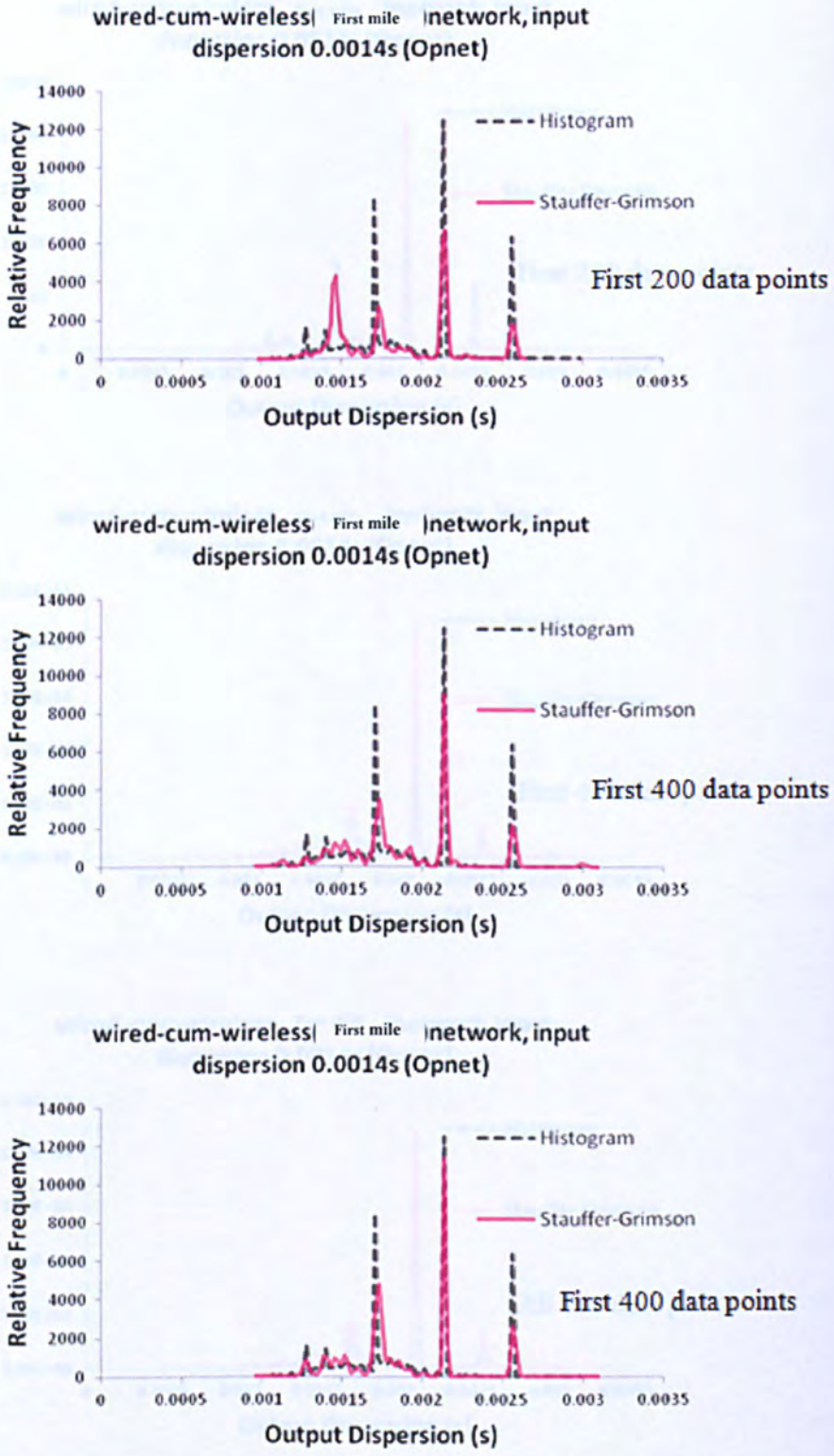
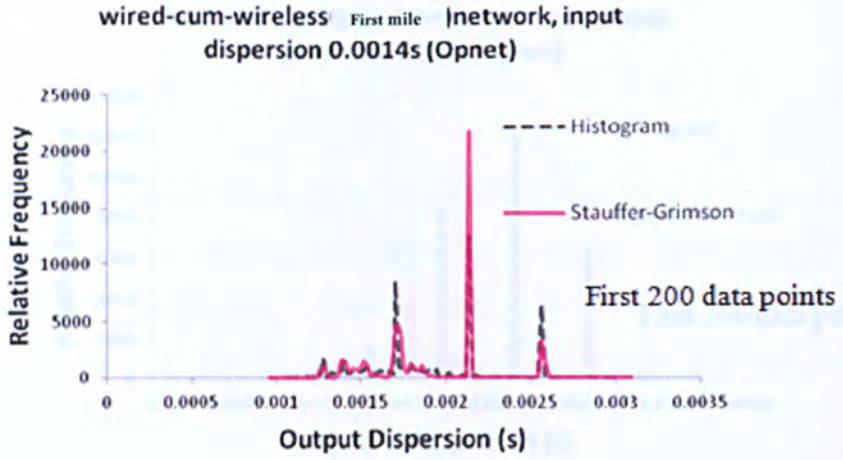
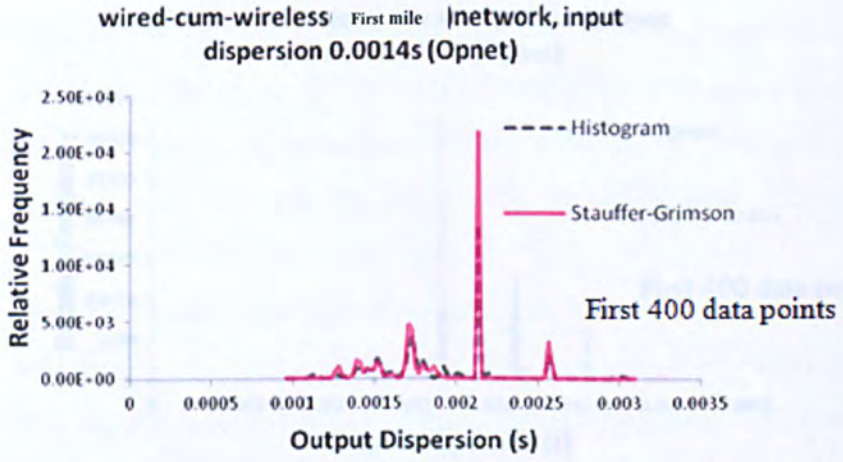


Figure 6.14 Stauffer-Grimson estimation ("slow" learning rate, α =0.01) of the dispersion distributions from the wired-cum-wireless network compared with corresponding histograms.





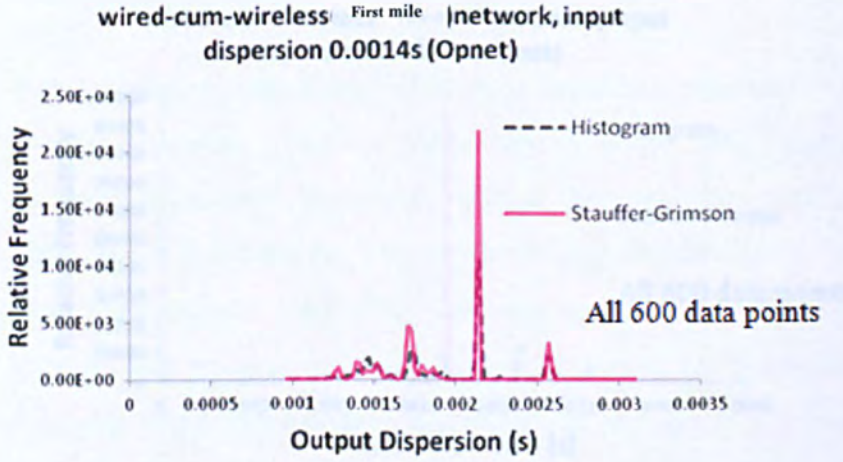


Figure 6.15 Stauffer-Grimson estimation ("medium" learning, α =0.02) of the dispersion distributions from the wired-cum-wireless network compared with corresponding histograms.

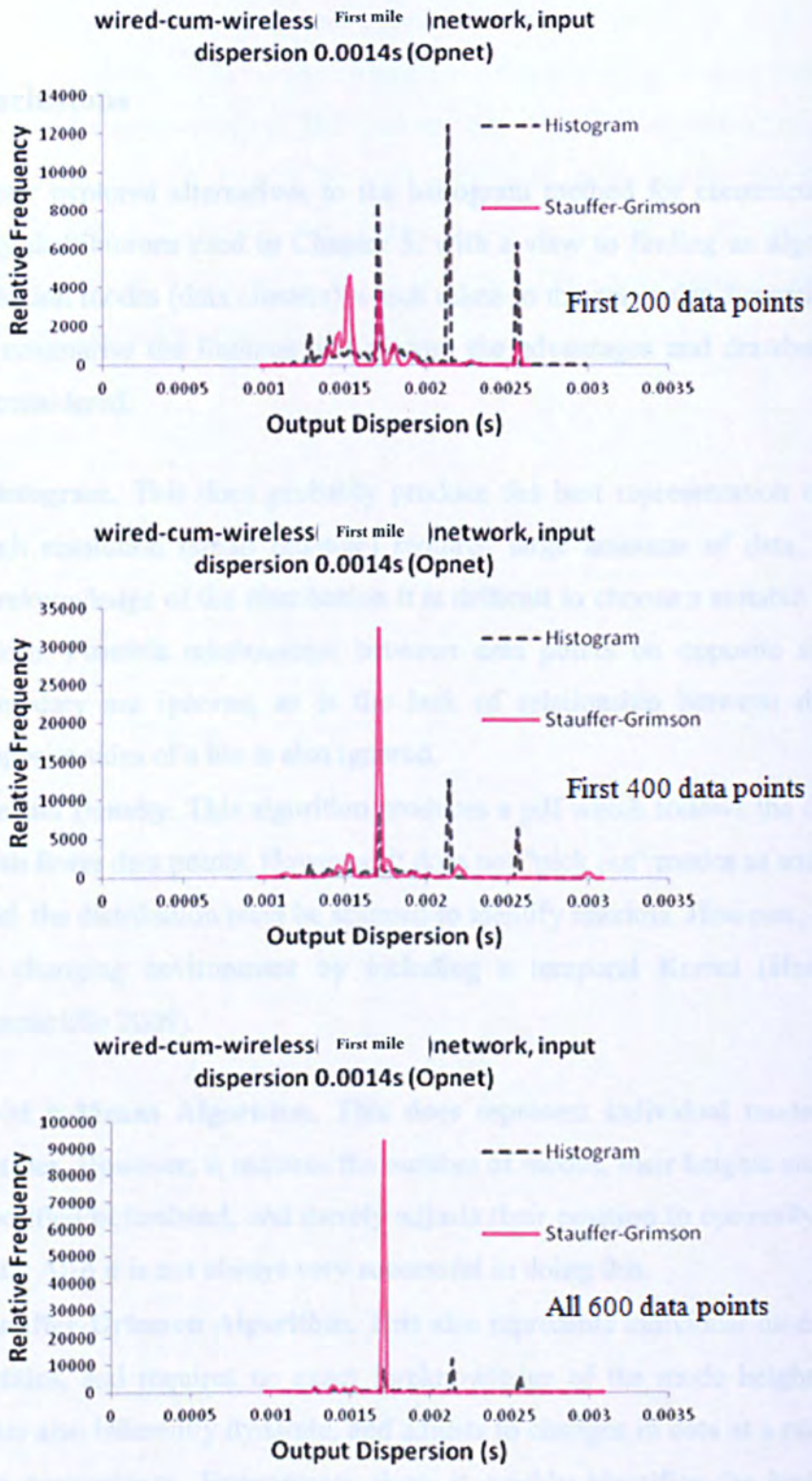


Figure 6.16 Stauffer-Grimson estimation ("fast" learning, α =0.1) of the dispersion distributions from the wired-cum-wireless network compared with corresponding histograms.

6.5. Conclusions

This chapter explored alternatives to the histogram method for constructing dispersion probability distributions used in Chapter 5, with a view to finding an algorithm to pick out distribution modes (data clusters) which relate to the particular dispersion signatures. Here we summarise the findings and discuss the advantages and drawbacks of all the methods considered.

- **Histogram.** This does probably produce the best representation of the pdf but high resolution (small bin-size) requires large amounts of data. Also without foreknowledge of the distribution it is difficult to choose a suitable range and bin width. Possible relationships between data points on opposite sides of a bin boundary are ignored, as is the lack of relationship between data points on opposite sides of a bin is also ignored.
- **Kernel Density**. This algorithm produces a pdf which follows the data quiet well with fewer data points. However, it does not "pick out" modes as software entities. and the distribution must be scanned to identify maxima. However, it can respond to changing environment by including a temporal Kernel (Hosseinpour and Tunnicliffe 2009).

- **E-M k-Means Algorithm.** This does represent individual modes as software entities. However, it requires the number of modes, their heights and widths to be specified beforehand, and merely adjusts their position to optimally represent the data. Also it is not always very successful in doing this.
- **Stauffer-Grimson Algorithm.** This also represents individual modes as software entities, and requires no exact foreknowledge of the mode heights and widths. This also inherently dynamic, and adjusts to changes in data at a rate specified by the parameter α. Experiments show it quickly identifies the histogram modes accurately, and it appears to work best with a medium learning rate (α=0.02). As well as the major signature modes it produces lots of little modes representing the independence noise and noise from random back offs. However, these could potentially be filtered.

Although the Kernel Density method addresses some of the issues with the histogram technique, it provides no direct representation of signature modes as software entities necessary for the automation of the probing process. Of the two other methods, the Stauffer-Grimson was found to be the more successful at representing the features of the distribution. It was therefore decided, based on the experience of the investigation, that the Stauffer-Grimson algorithm is the best option. The remainder of this thesis will therefore use the Stauffer-Grimson algorithm.

7. DEVELOPING A CLOSED LOOP PROBING ALGORITHM

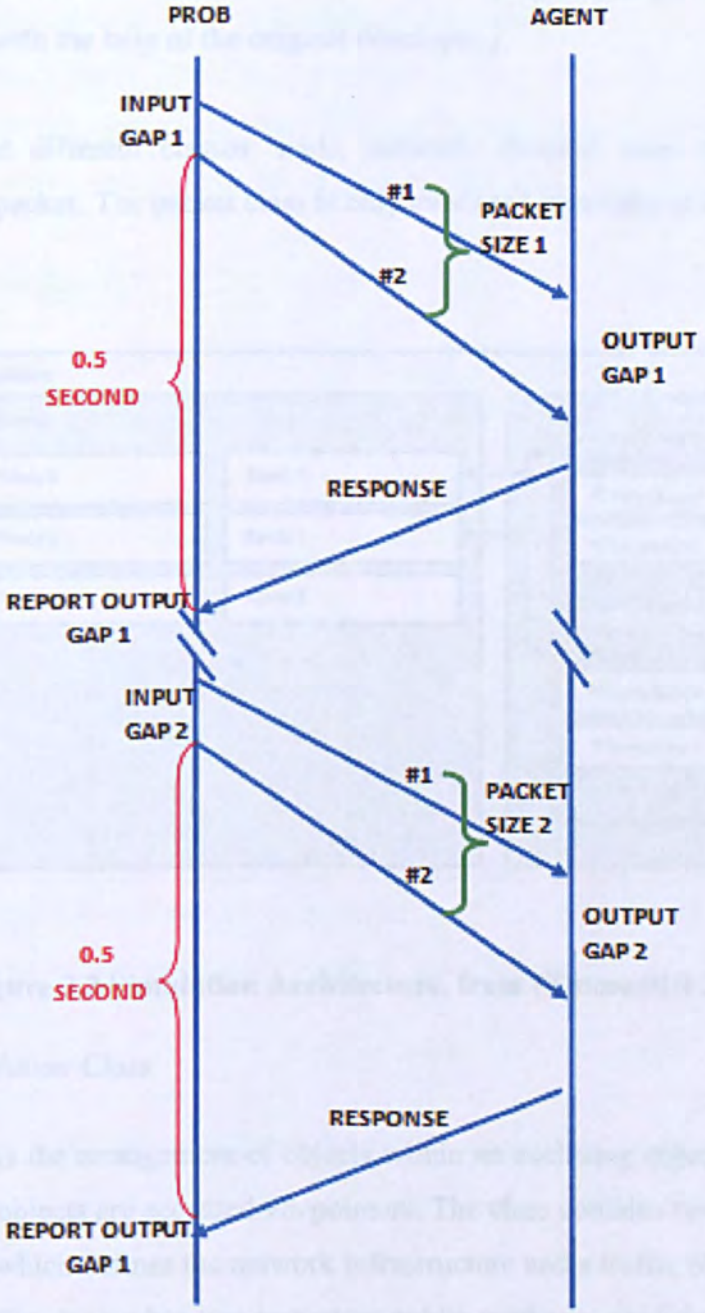
7.1. Introduction

The past three chapters have used Opnet as a tool for investigating packet-pair probing in wired and wired-cum-wireless networks. Chapter 4 outlined the development of the packet pair/stream modules and investigated the baseline scenario of two workstations communicating directly on a wired or wireless channel. It was shown how in this simple arrangement using two different probe packet sizes, the link parameters (bandwidth and the inter-packet gap) obtained from the packet pair measurements compared well with their known values. Chapter 5 extended this work to larger networks of nodes carrying cross-traffic, giving rise to multimodal dispersion distributions, which were plotted using the histogram method. The modes correspond to probing signatures identified by (Pásztor and Veitch 2002) and the distributions heavily depend on the input gaps. Chapter 6 investigated alternatives to the histogram method, and showed how nodes can be automatically classified as discrete software objects using the algorithm of (Stauffer and Grimson 2000).

The current chapter builds further upon these investigations by bringing the network and the data classification into a common simulation framework, allowing the probe process to classify dispersion dynamically during operation, and adjust its input-gap for optimum measurement. Figure 7.1 shows the proposed scheme: an agent process on the path sink measures the output dispersion and returns this information to the probe source for analysis. Two different probe packet sizes are transmitted alternately, such that the effects of bandwidth and header/inter-frame gap can be distinguished (see Chapter 4).

For this purpose we moved away from Opnet to an alternative simulator called ProbSim, developed at Kingston University by (Tunnicliffe 2010). ProbeSim (like ns-2) consists of a collection of C++ classes which can be used to create network probing simulations. It is

easily modified, considerably simpler than Opnet, and the expertise to adapt its code was readily available.



7.1 Timing diagram for probe/agent process

7.2. The ProbeSim Simulation Software

ProbeSim consists of a series of C++ classes which can be downloaded from a URL (http://staffnet.kingston.ac.uk/~ku12881/netclasses/) and users can modify the source

code for their own purposes. Results are generated in the form of text files which can be analyzed by using several different tools, most commonly Microsoft Excel. (The development of ProbeSim itself was not part of this project, though some of the classes were modified with the help of the original developer.)

There are eight different classes: node, network, channel, user, connection, probe, simulation and packet. The packet class is only ever used internally to represent data units in the network.

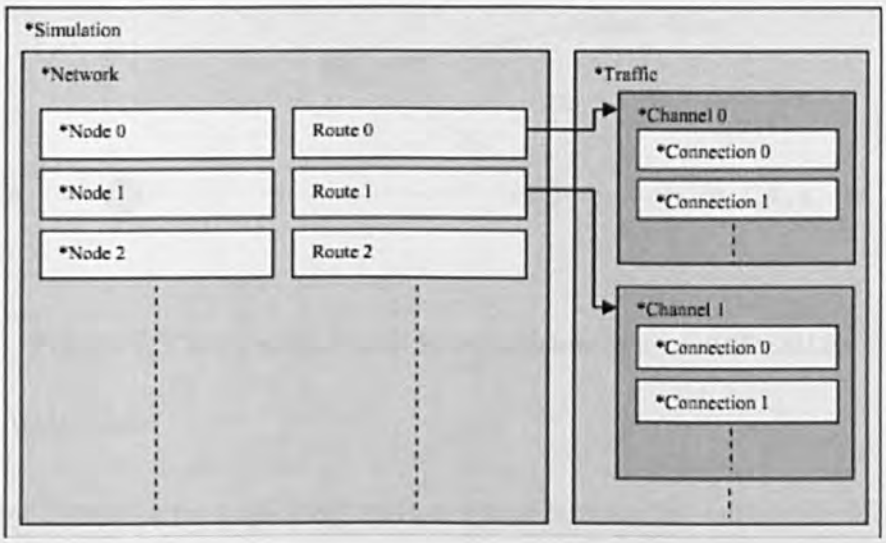


Figure 7.2 Simulation Architecture, from (Tunnicliffe 2010).

7.2.1. The Simulation Class

Figure 7.1 shows the arrangement of objects within an enclosing object of the simulation class. All these objects are accessed via pointers. The class contains two main attributes: a network object which defines the network infrastructure and a traffic object which defines the network traffic. It also has two important public methods: configure() which supplies it with a network infrastructure and traffic profile and advance() which progresses the simulation through a specified number of seconds. It also contains a method add_jitter() which can be used to introduce random Gaussian timing jitter between network endpoints. In these experiments this was set to 1ns.

7.2.2. The Network Class

Objects of this class specify network nodes and the routes between them. When specifying a route, objects of the node class are identified by integers, and -1 indicates a network sink (see Figure 7.3). These numbers indicate the order in which they were added to the network object. The class has four important public methods and they are insert_node(), insert_ route(), get_ no_of_nodes() and get_ no_of_routes().

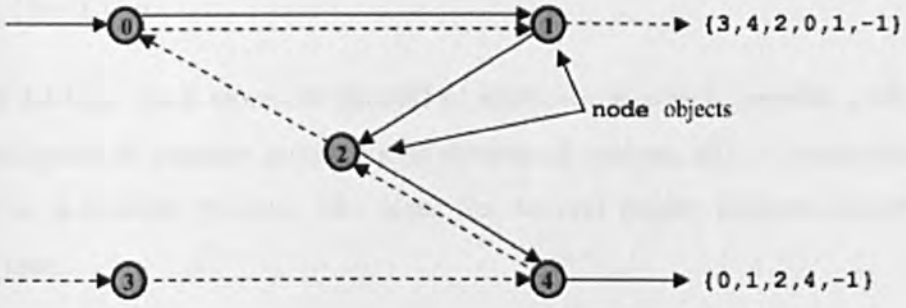


Figure 7.3 Typical Network Routes, from (Tunnicliffe 2010).

7.2.3. The Node Class

A Node object represents a queuing system which mimics the behaviour of a drop-tail router or store-and-forward switch. It processes data at a specified bit-rate, and imposes an inter-frame gap, which may have a constant or random size.

7.2.4. The Traffic Class

An object of the traffic class specifies the traffic applied to the network. It contains a series of Channel objects with integer identifiers 0,1,2..., which are added using the method insert_channel().

7.2.5. The Channel Class

The traffic applied to each route within a network is specified by a Channel object, which is a container for several independent connections which specify the actual transmission or traffic processes, and are added to the channel using the method insert_connection().

7.2.6. The Connection Class

Objects of this class contain one of three attributes: a User, a Probe or an Agent. A User is responsible for specifying user-related traffic process, a Probe which is responsible for measuring the bandwidth and an Agent is responsible for returning probe information to its source for analysis. They are added to the object using the methods insert_user(), insert_probe() and insert_agent().

7.2.7. The User Class

Objects of the User class represent networked applications which generate packets. They can be configured to generate a continuous streams of packets, either deterministically or according to a random process. The class has several public methods which will be discussed later.

7.2.8. The Probe Class

Bandwidth monitoring algorithms are implemented in objects of the Probe class, and apply to the connection into which the Probe object is inserted. There are a number of methods of this class which will be discussed later.

7.2.9. The Agent Class

An additional class needed to represent the sink process at the end of a probed path, which returns the probing information to its source for analysis. Every agent object is associated with a specified Probe object, to which it returns its response packets. These response packets carry the input "long gap" (between "first" packets of the same size) the input "short gap" (between first and last packets of a pair/stream and the corresponding output long and short gaps. (This is illustrated in Figure 7.1.)

7.3. Design and Verification

The following five simulation models were used:

- Baseline wired (Figure 7.4)
- Baseline wireless (Figure 7.6)
- Network wired (Figure 7.8)

- Network wireless "firstmile" (Figure 7.11)
- Network wireless "lastmile" (Figure 7.14)

These are identical to the corresponding models used in Chapters 4 and 5, with the exception that they include a "return path" for the agent response packets. However, since ProbeSim is not a recognized industry standard, the first task was to verify its operation against Opnet.

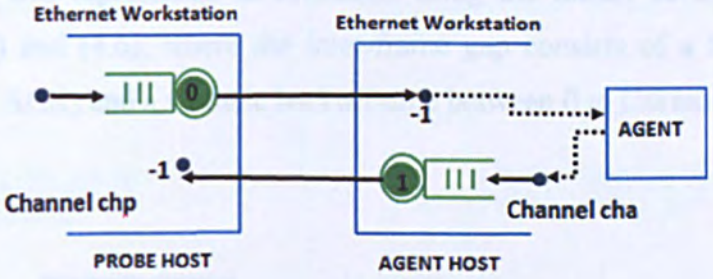


Figure 7.4 Wired Baseline model

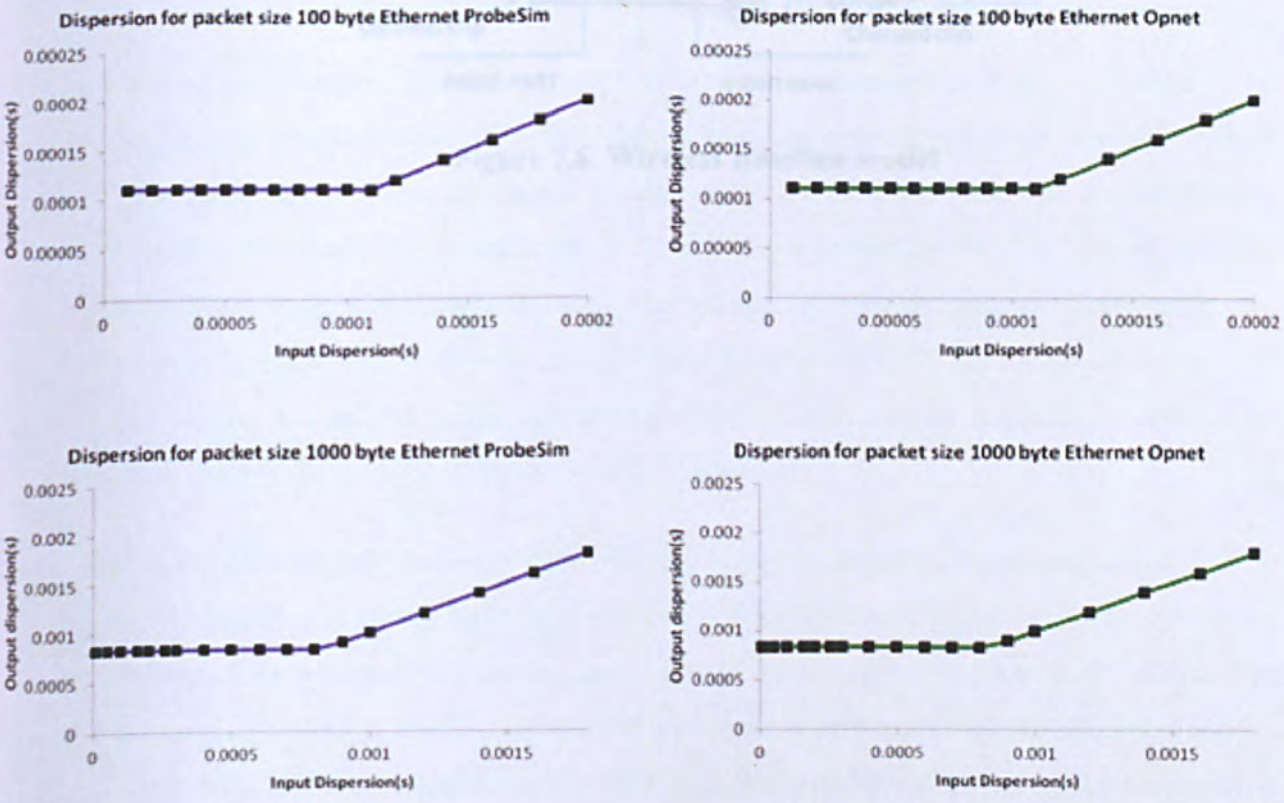


Figure 7.5 Wired Baseline results from ProbeSim compared with Opnet

A wired Ethernet node was modelled by setting the bit rate equal to 10Mbit/s and the inter-frame gap to a constant 38 byte times. Figure 7.5 shows the results of a "baseline" wired experiment of two wired 10BaseT workstations directly connected with no intervening network and no cross-traffic, for packet sizes 100 and 1000 bytes. The results are indistinguishable, showing that in this scenario the ProbeSim produces results as good as the commercial simulator.

A wireless node was represented in ProbeSim using the model developed in Section 4.4.3, Eqns. (4.5) and (4.6), where the inter-frame gap consists of a fixed gap (SIFS, DIFS and time to ACK) and a variable backoff time between 0 to CWmax.

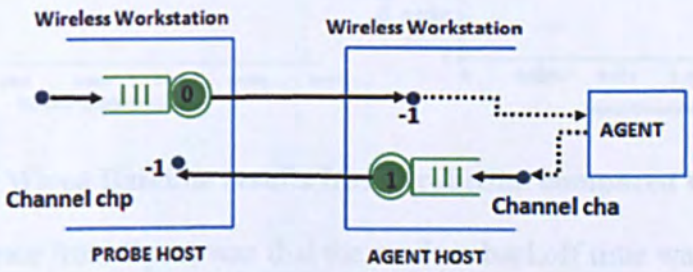


Figure 7.6 Wireless Baseline model

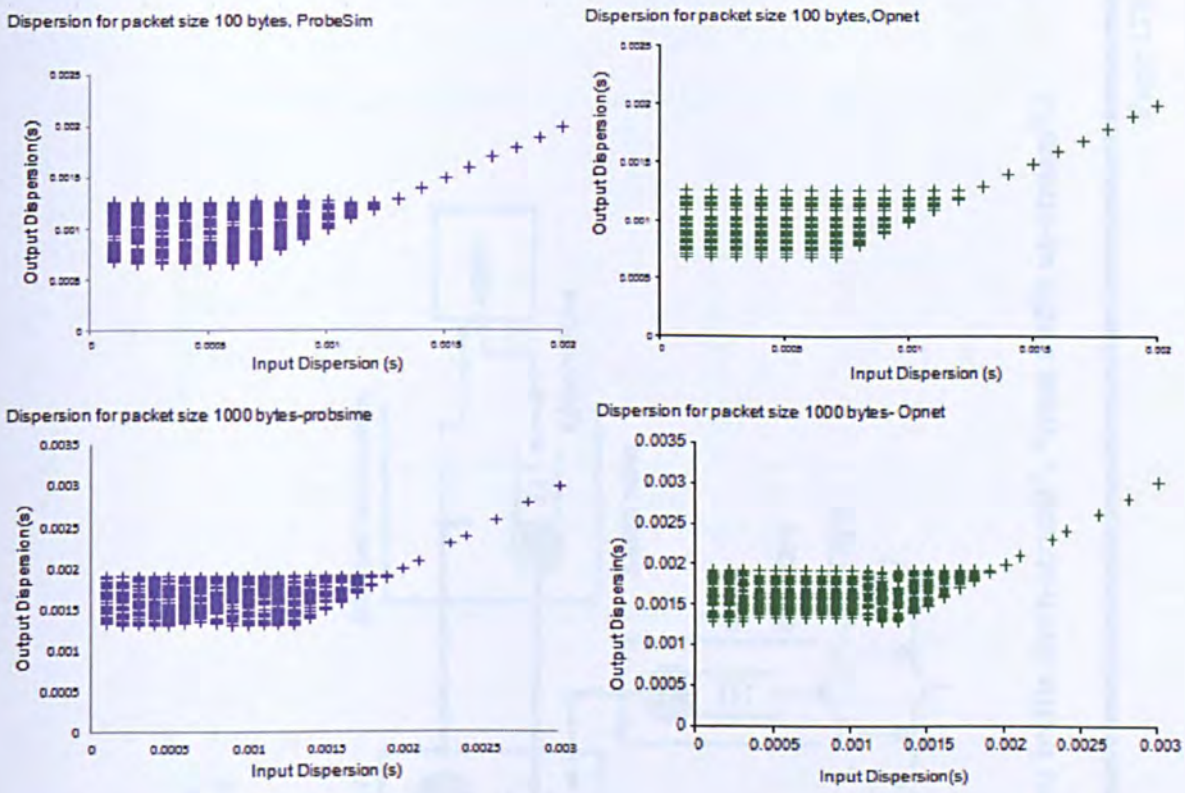


Figure 7.7 Wired Baseline results from ProbeSim compared with Opnet

One slight difference from Opnet was that the random backoff time was represented as a continuous random variable rather than a discrete number of slots, but the overall simulation results were not found to be very greatly affected by this approximation. Figure 7.5 shows the results of a "baseline" experiment of two wired wireless workstations directly connected with no intervening network and no cross-traffic, for packet sizes 100 and 1000 bytes. Aside from the granularity in the Opnet backoff times, the results are almost indistinguishable, showing that in this scenario the ProbeSim produces results as good as the commercial simulator.

The experiments of Chapter 6 were repeated using ProbeSim, again using a cross-traffic traffic packet size of 500 bytes, and the results compared with Opnet. The two simulators produced an impressively close agreement (Figures 7.8-7.15), the small differences between them being mostly attributable to statistical differences in the simulated traffic. This result was to be expected since ProbeSim was configured to mimic the link models described in Section 4.4 which were themselves based on the Opnet link models. ProbeSim can therefore be considered a valid tool for this area of research.

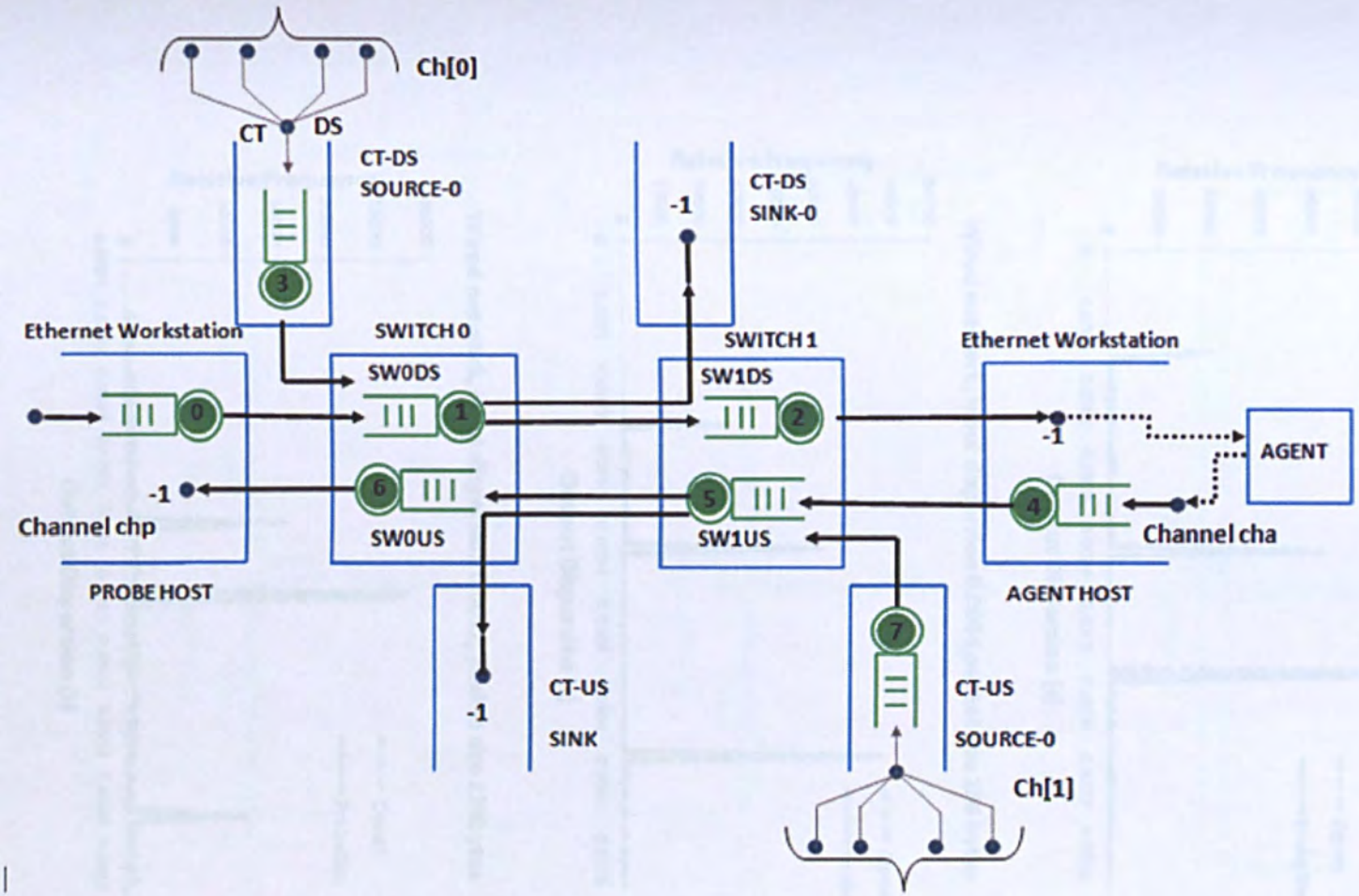


Figure 7.8 Wired ProbeSim Model (CT-DS, CT-US etc. stand for "cross traffic down-stream", "cross traffic up-stream".)

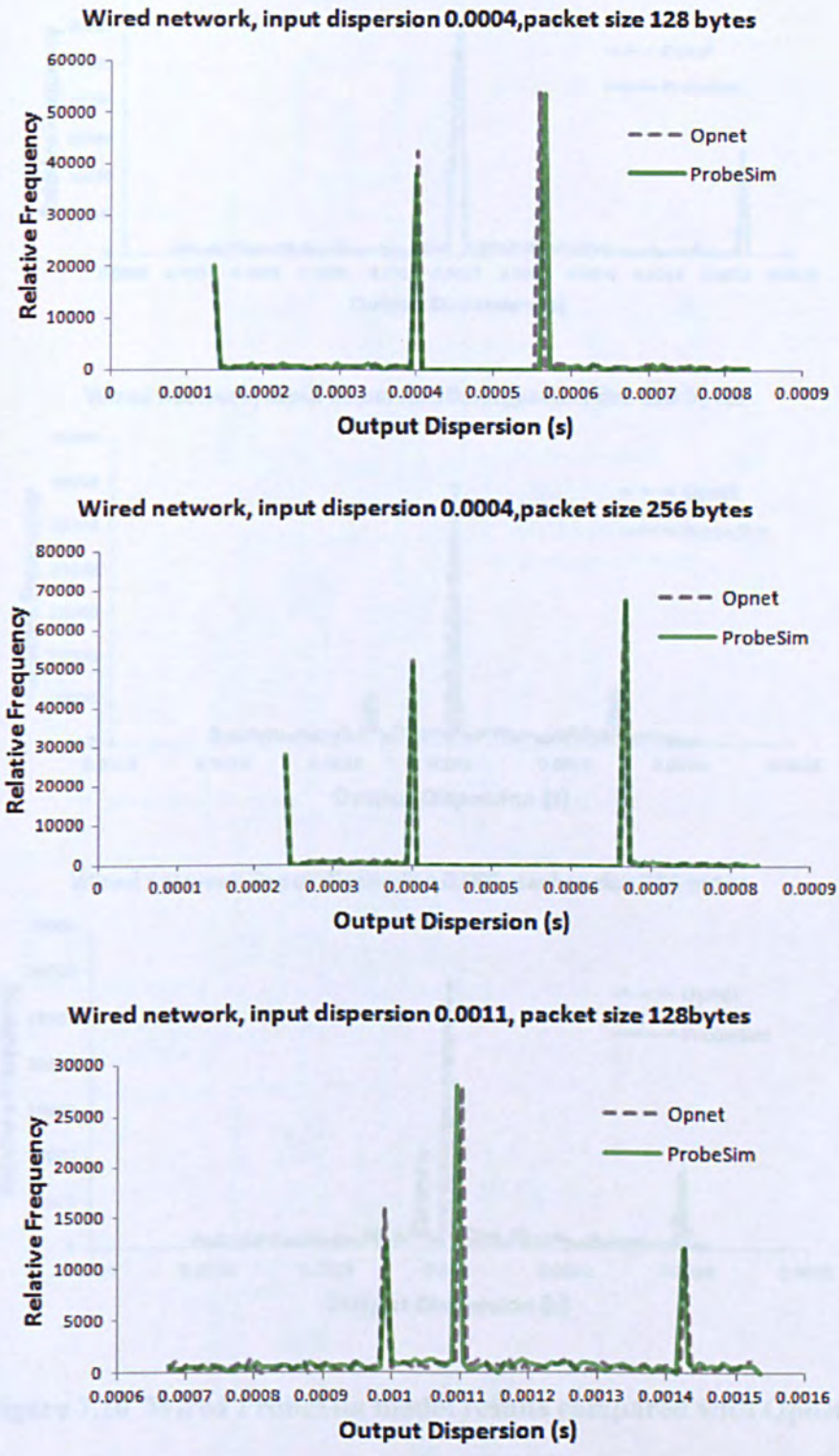


Figure 7.9 Wired ProbeSim model results compared with Opnet

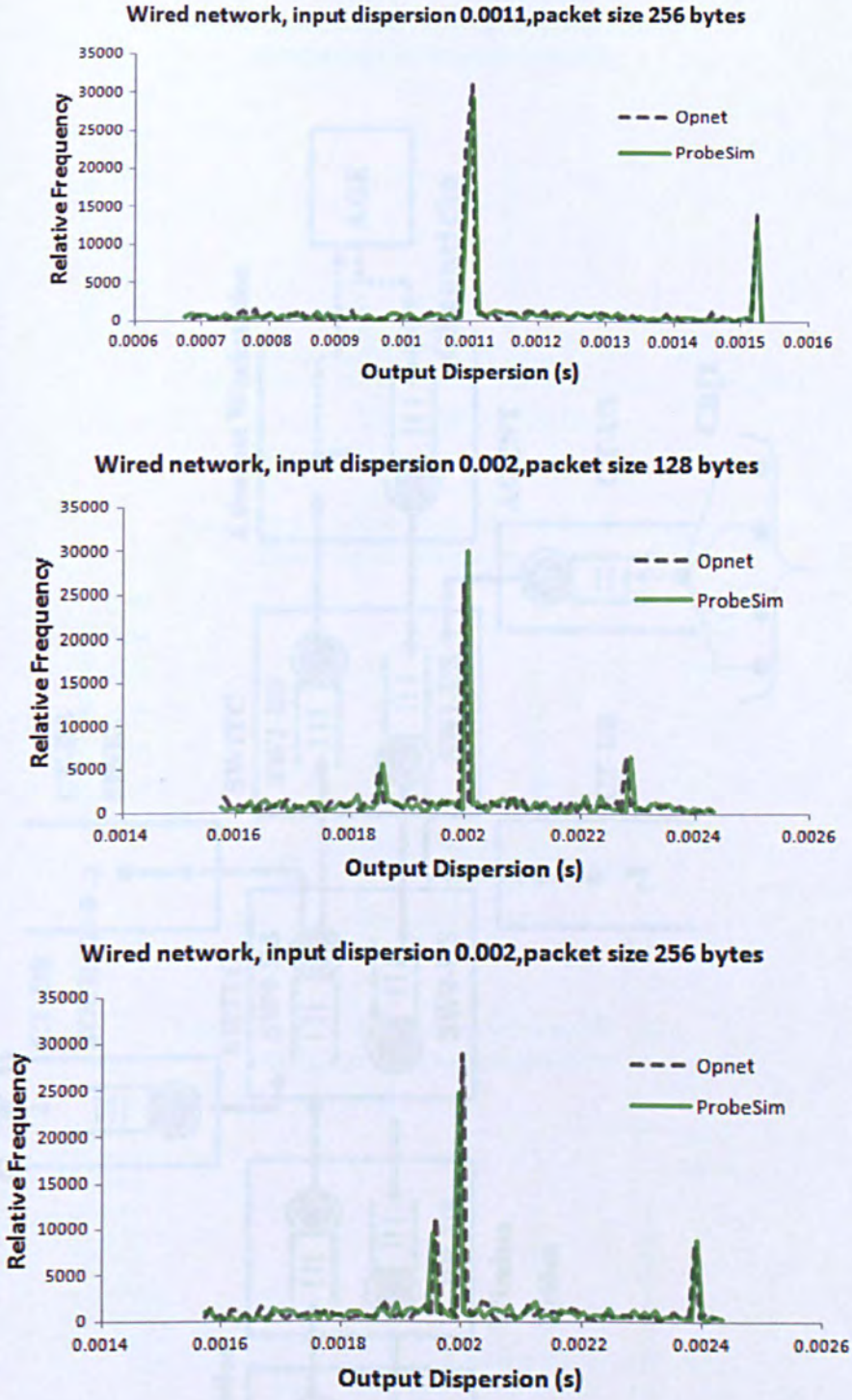


Figure 7.10 Wired ProbeSim model results compared with Opnet

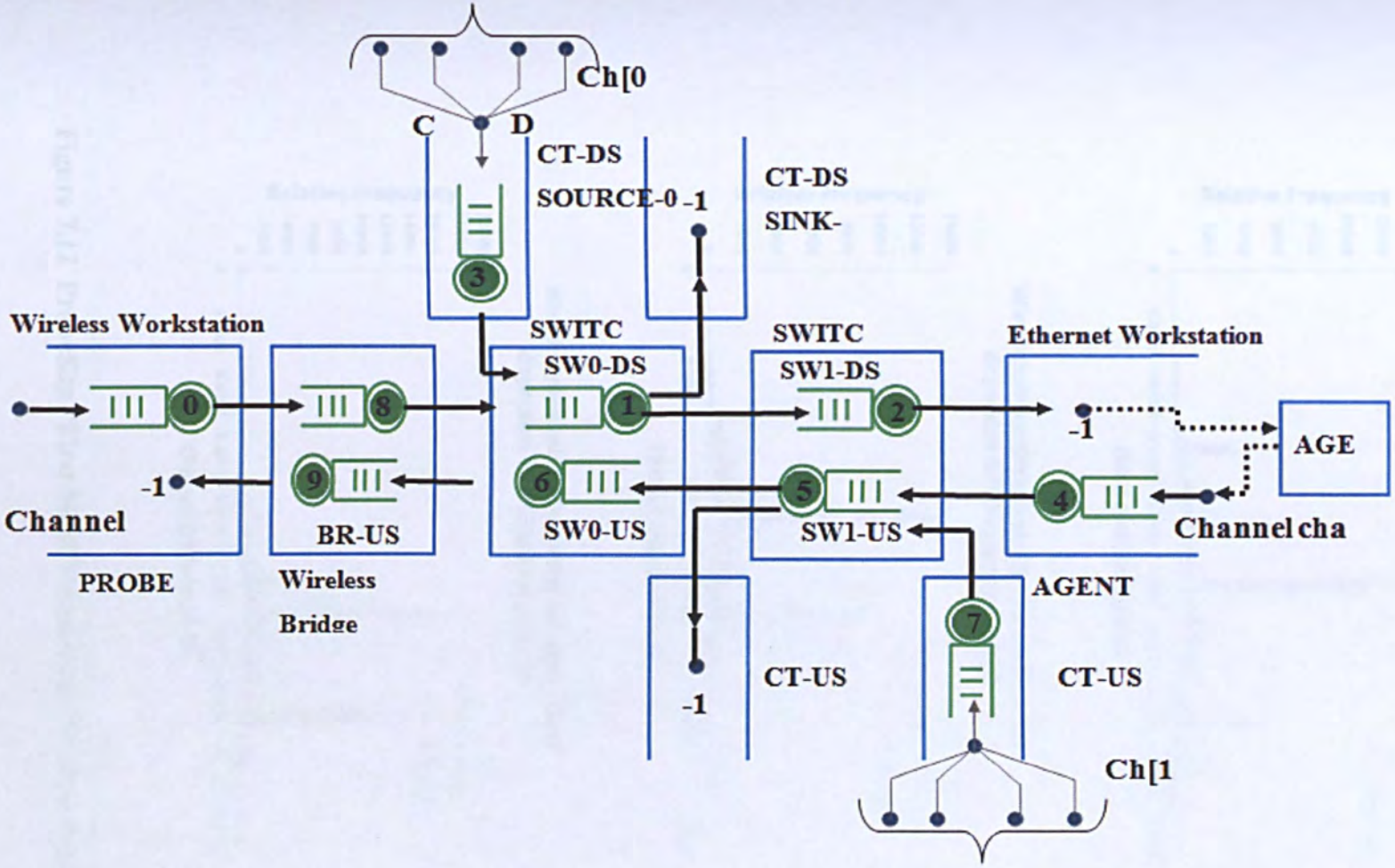
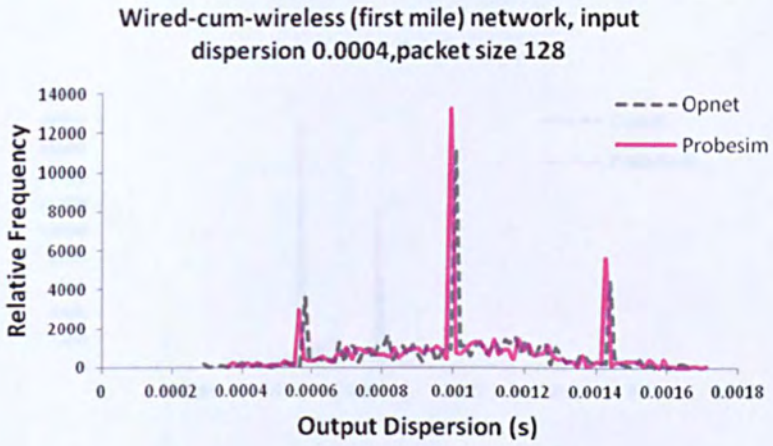
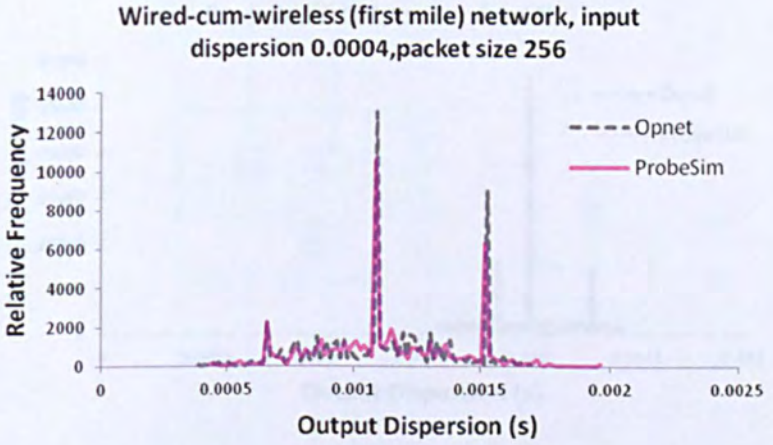


Figure 7.11 ProbeSim "First Mile"Wired-cum-Wireless network model





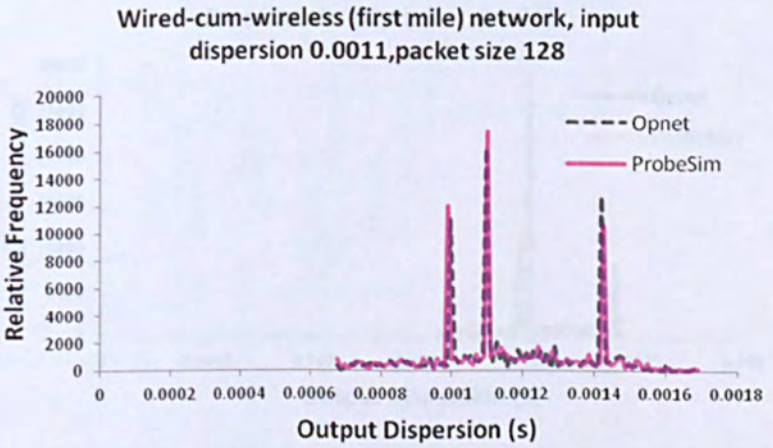


Figure 7.12 ProbeSim "First Mile"Wired-cum-Wireless results

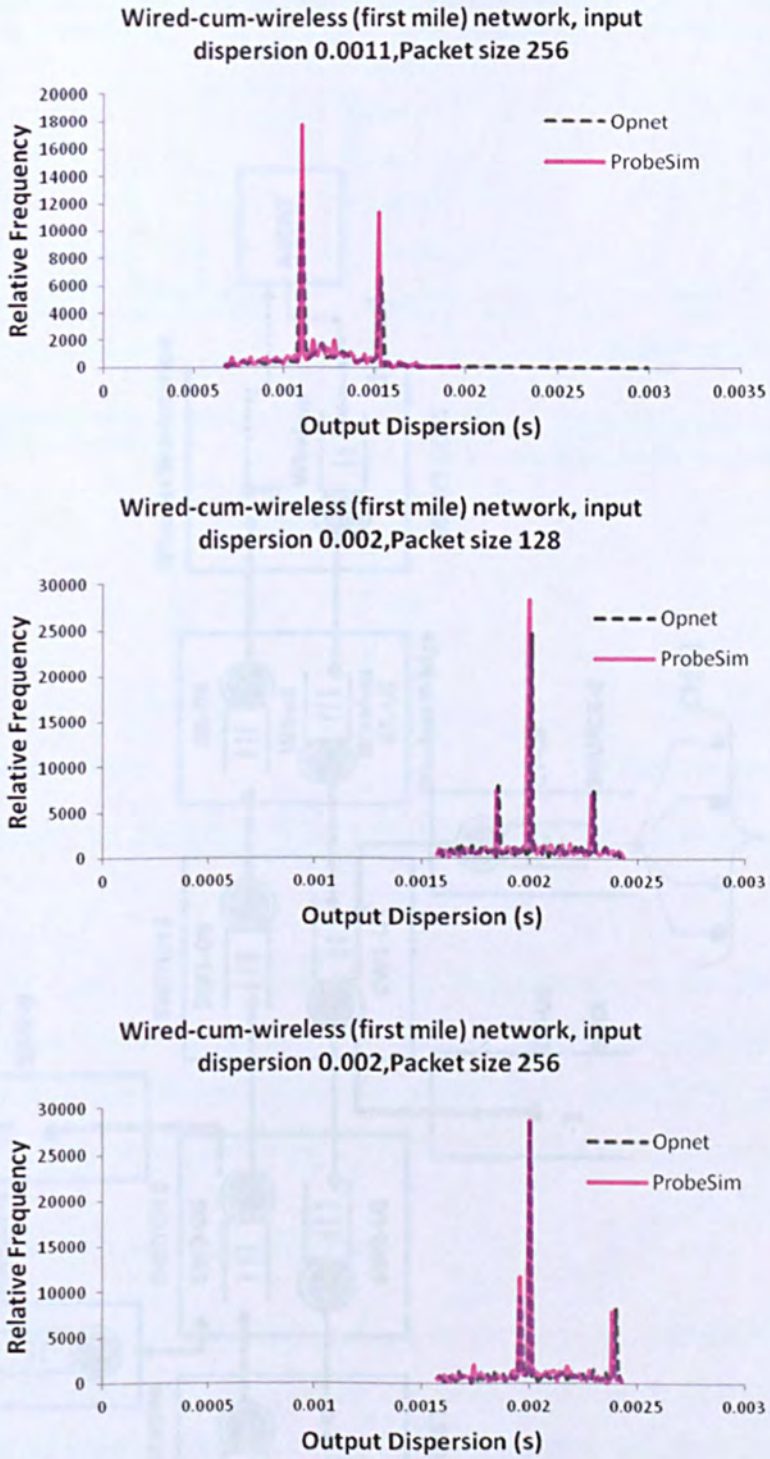


Figure 7.13 ProbeSim "First Mile"Wired-cum-Wireless results.

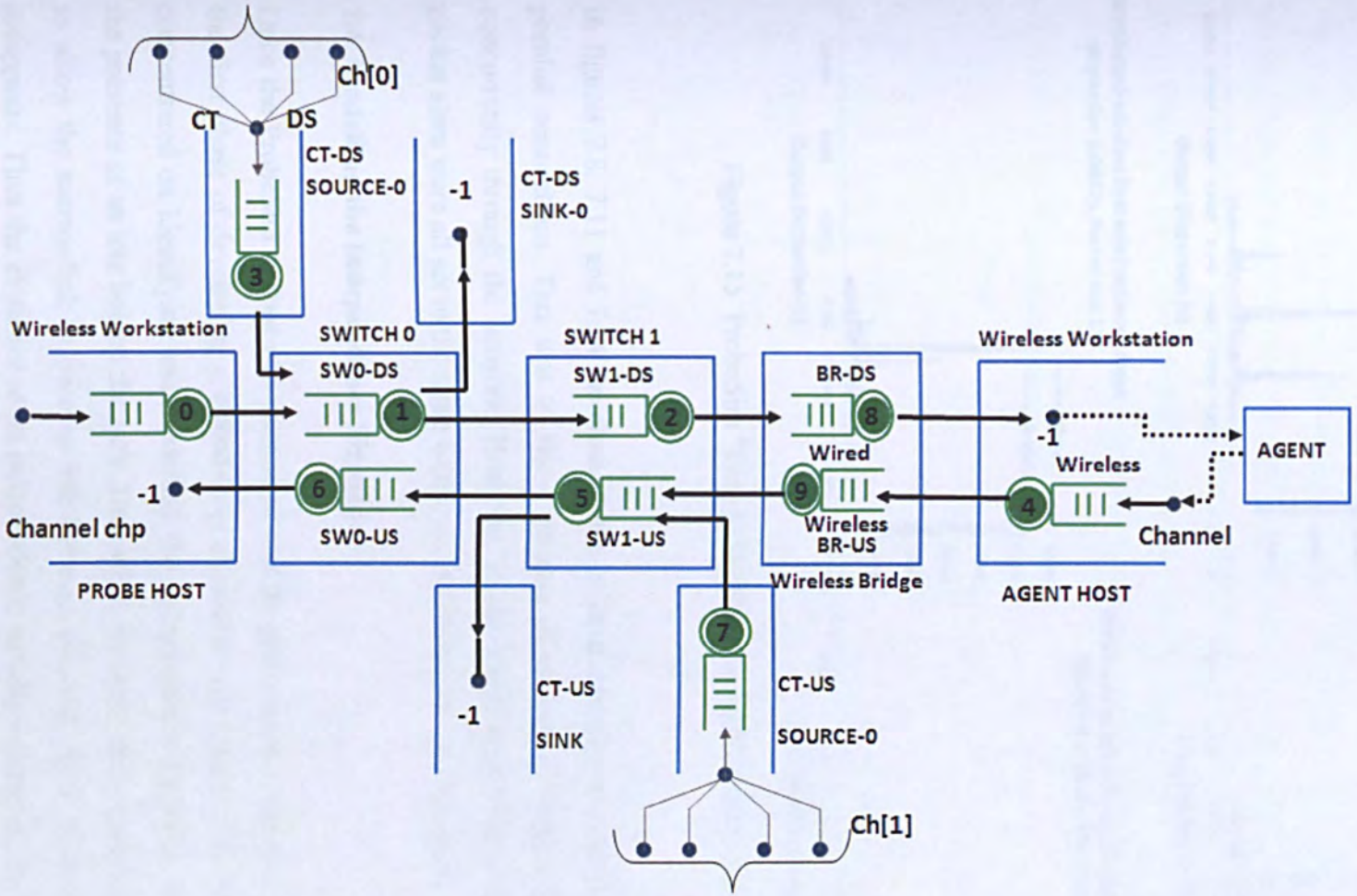


Figure 7.14 ProbeSim "Last Mile "Wired-cum-Wireless network model

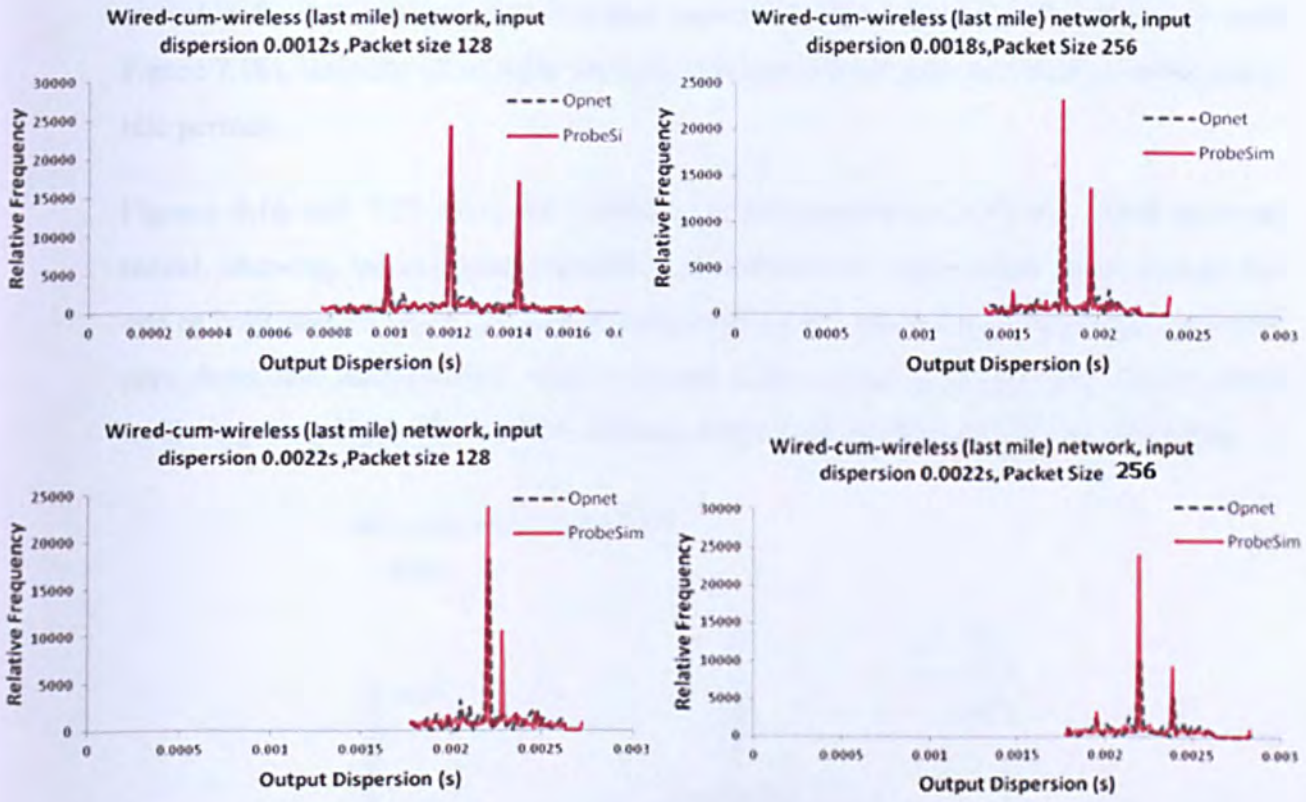


Figure 7.15 ProbeSim "First Mile" Wired-cum-Wireless results

In figures 7.8, 7.11 and 7.14 the cross-traffic in each direction is actually made of four parallel connections. This was to allow streams of packets of different sizes to flow concurrently through the network. However, in the above experiments the cross traffic packet sizes were all set to the same value, i.e. 500 bytes (the *ns-2* default).

7.4. Modelling the Independence Signature

Once the ProbeSim testbed was completed and its performance validated against Opnet, the first phase of developing a closed-loop algorithm was begun. The first experiments concentrated on identifying and modelling the independence signature, which indicated the presence of an idle link on the path. Thus when the input probe separation is too small to allow the narrow-link to become idle between packets, the independence signature disappears. Thus the existence of an independence signature indicates that the probe rate is below the narrow link bandwidth. In the independence signature $\Delta_{out} \approx \Delta_{in}$, but the finite random differences between these quantities (due to timing jitter, which may vary during an experiment due to environmental factors). This information was readily

available from the "long gaps" between successive first packets of the same size (see Figure 7.18), since for all sensible levels of utilization these gaps inevitably contain many idle periods.

Figures 4.16 and 7.17 show the results of some experiments with the wired network model, showing independence signatures surrounded by independent noise (causes by one or both packets being delayed by cross-traffic). It is clear that independent signature goes down and independence noise increases with increasing packet size. As the cross traffic is getting bigger the independent signature is getting smaller and less noticeable.

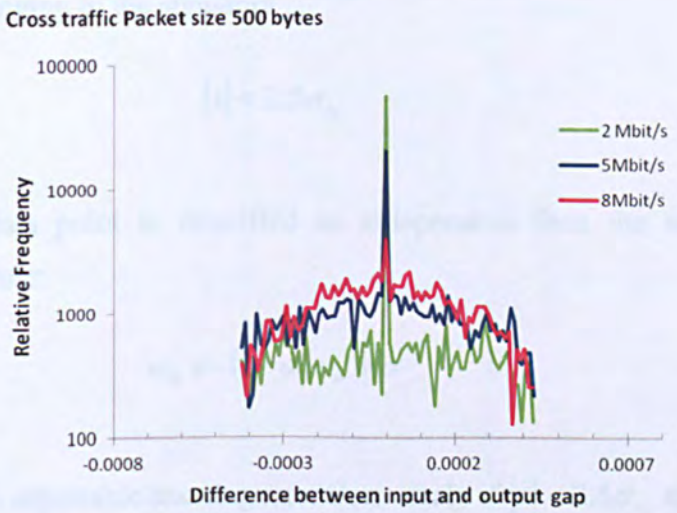


Figure 7.16 Independence signature and noise for 500 byte cross-traffic.

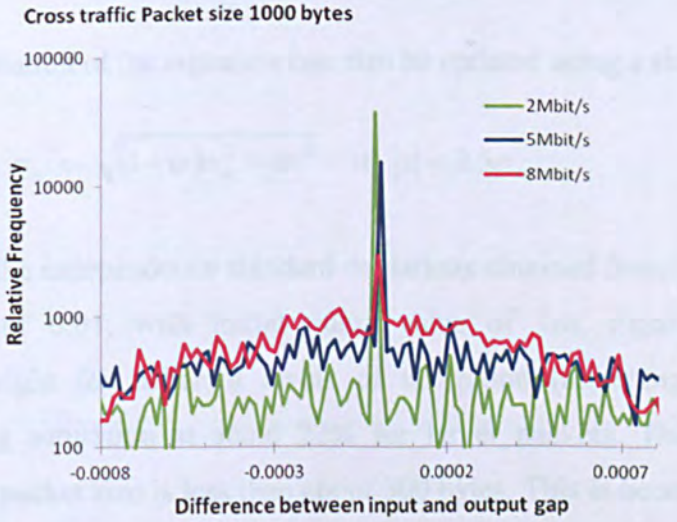


Figure 7.17 Independence signature and noise for 1000 byte cross traffic.

Although the clock jitter has been set in the simulation, the probing process may not know its value and therefore needs to process this information in order to measure it. We use a similar method to the Stauffer-Grimson algorithm investigated in Chapter 6, but with only one single mode to represent the independence signature:

$$x = \Delta^*_{out} - \Delta^*_{in} \tag{7.1}$$

where the asterisks indicate the long input and output gaps. We model the independence signature modelled in terms of a weight w_{is} and a standard deviation σ_{is} and classify a data point as belonging to the signature

$$|x| < 2.5\sigma_{is} \tag{7.2}$$

If n incoming data point is classified as independent then the weight is increased according to the rule:

$$w_{is} \leftarrow (1-\alpha)w_{is} + \alpha \tag{7.3}$$

Where alpha is an adjustable learning rate. Conversely if $|x| < 2.5\sigma_{is}$ then

$$w_{is} \leftarrow (1-\alpha)w_{is} \tag{7.4}$$

.The standard deviation of the signature can also be updated using a similar learning rate.

$$\sigma_{is} \leftarrow \sqrt{(1-\alpha)\sigma_{is}^2 + \alpha x^2} \quad \text{if } |x| < 2.5\sigma_{is} \tag{7.5}$$

Table 7.1 shows the independence standard deviations obtained from long gap data using a learning rate of 0.01, with initial guess value of 1ns. Figure 7.19 shows the independence weight for different levels of utilization (assuming initial weight of 0.00001) showing saturation at about 22% for larger packets. The weight decreases rapidly when the packet size is less than about 300 bytes. This is because smaller packets imply a much greater *number* of packets, and hence a greater probability that the probe packets will be randomly delayed. Figure 7.20 shows how for larger packets these weights depend strongly on cross traffic rate and must more weakly upon packet size.

Independence signatures are extremely weak when the utilization is high, and in these regions the long-gap information is probably not so useful.

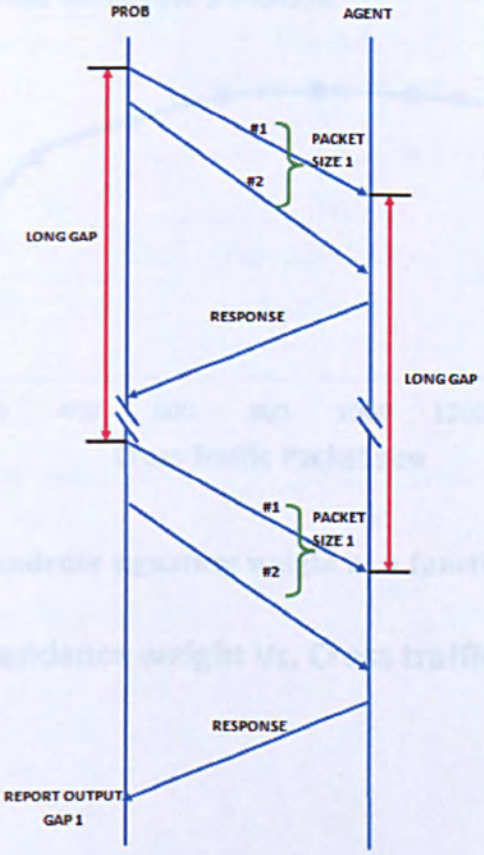


Figure 7.18 Timing diagram for probe and agent-response packets.

	2Mbit/s	5Mbit/s	8Mbit/s
100 bytes	1.43E-09	1.27E-09	1.00E-09
500 bytes	1.40E-09	1.15E-09	1.15E-09
1000 bytes	1.21E-09	1.39E-09	1.15E-09

Table 7.1 Sizes of independence mode produced from long-gap data.

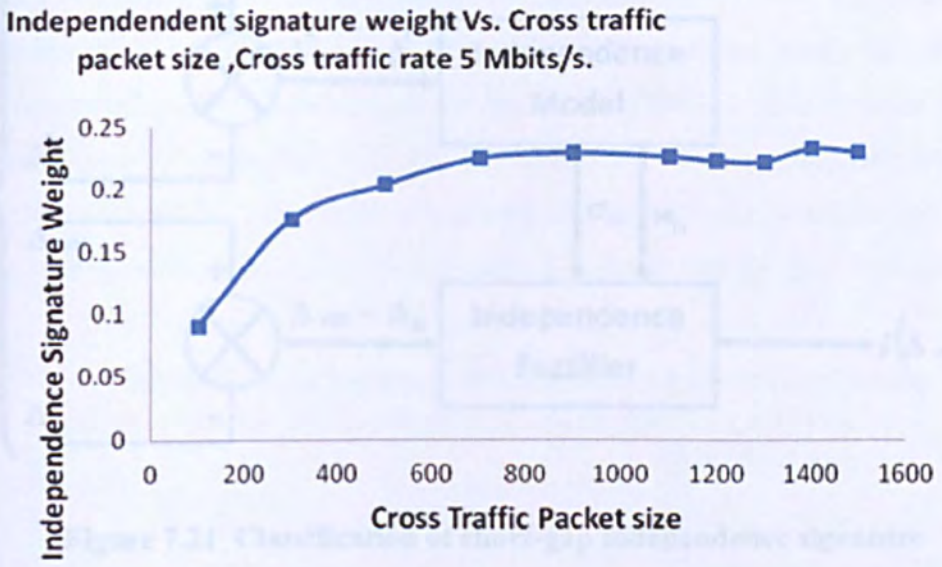


Figure 7.19 Independence signature weight as a function of packet size.

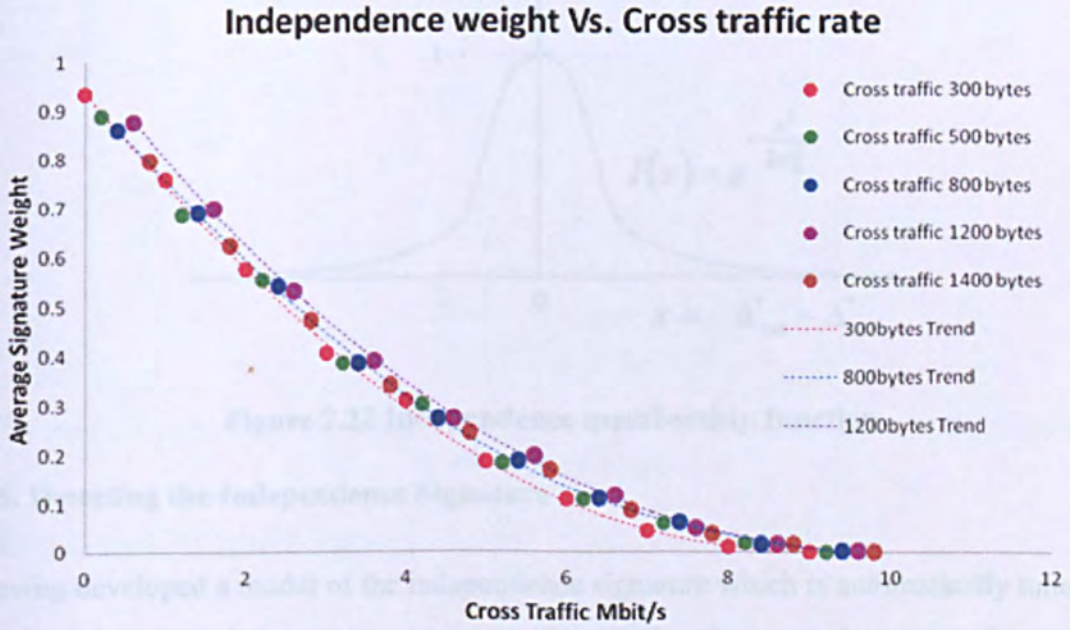


Figure 7.20 Independence weight as a function of cross traffic rate.

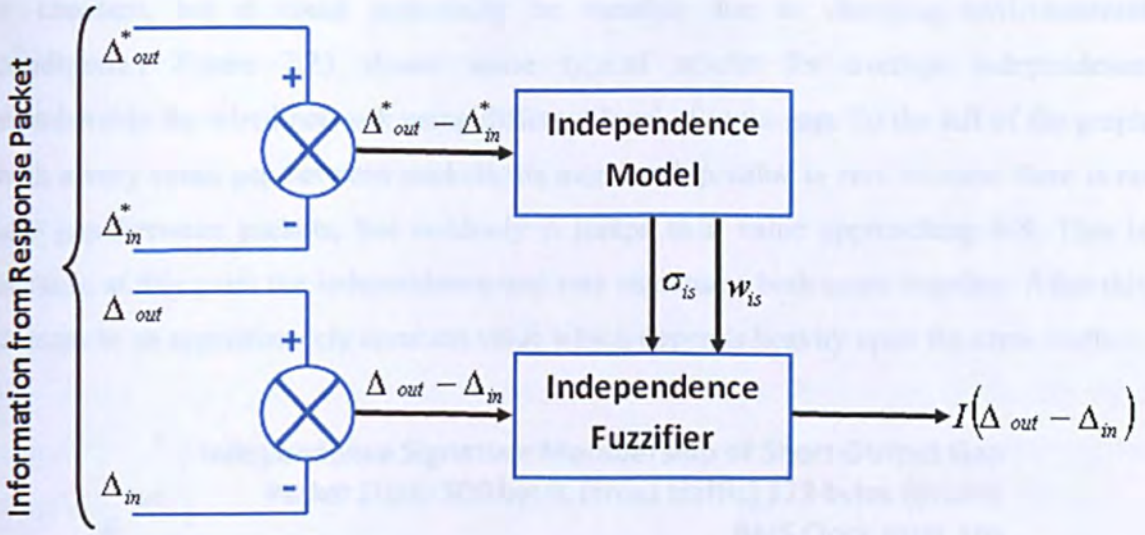


Figure 7.21 Classification of short-gap independence signature

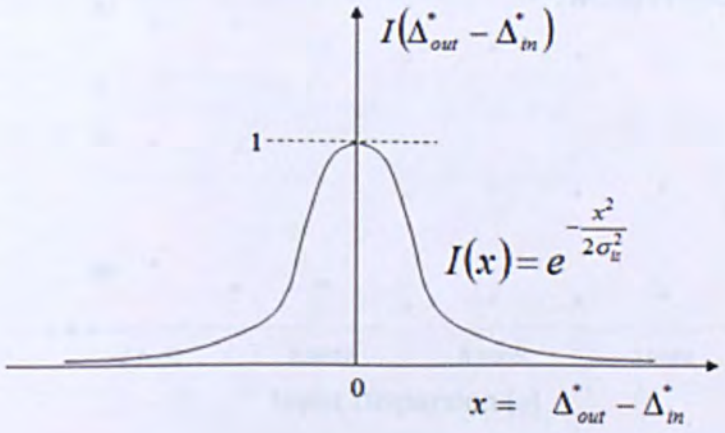


Figure 7.22 Independence membership function

7.5. Detecting the Independence Signature

Having developed a model of the independence signature which is automatically tuned by the long-gap data, it is now necessary to classify the short-gap data using this model. Figure 7.21 shows the architecture developed to do this: The independence model uses the long gap data (in which the packets are so far apart and the idle period between them can generally be assured) to tune a "fuzzifier", whose purpose is to assign a "fuzzy membership" level to each short-gap piece of data, signifying the degree to which it belongs to the independence signature (see Figure 7.22) at the time at which it was measured. The independence model dynamically tunes the fuzzifier to detect the independent signature in the short gap. (In this experiment the underlying value of sigma

is constant, but it could potentially be variable due to changing environmental conditions.) Figure 7.23 shows some typical results for average independence membership for wired network using different level of input gap. To the left of the graph with a very small gap between packets, its membership value is zero because there is no idle gap between packets, but suddenly it jumps to a value approaching 0.8. This is because at this point the independence and rate signatures both come together. After this it drops to an approximately constant value which depends heavily upon the cross traffic.

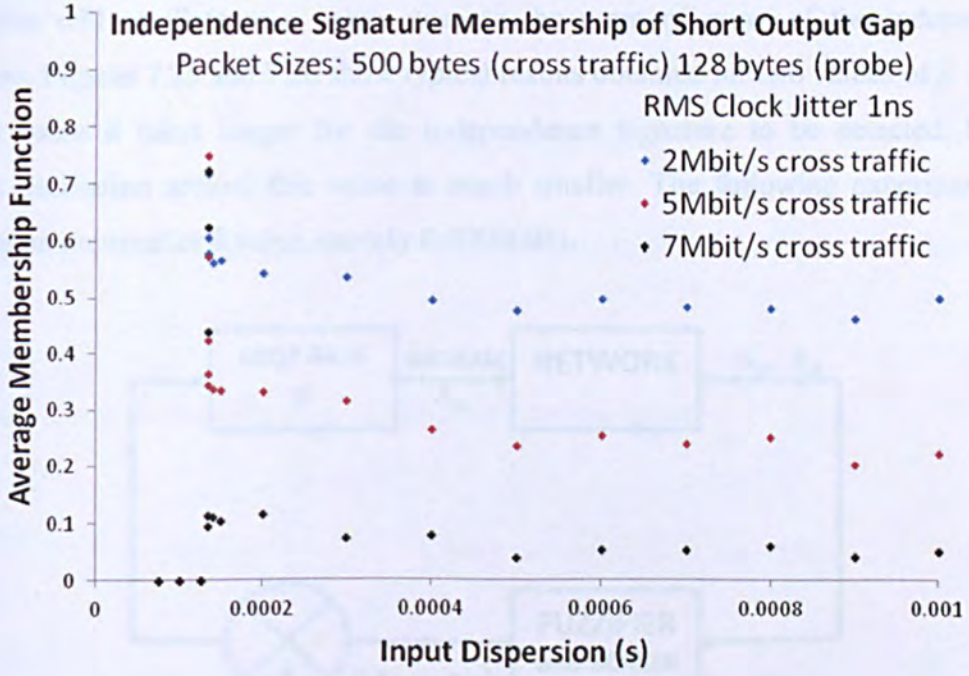


Figure 7.23 Average independence membership for wired network using different levels of input gap.

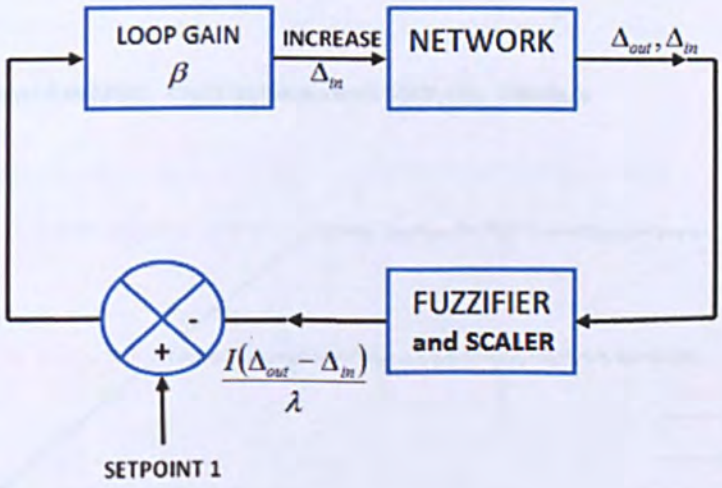
7.6. Control Algorithm

The following control algorithm was used to detect the onset of the independence signature. The input dispersion was initially set to a very small value (zero in this simulation) and altered at each measurement according to the rule:

$$\Delta_{in} \leftarrow \Delta_{in} + \beta\left[1 - \frac{I(\Delta_{out} - \Delta_{in})}{\lambda}\right] \tag{7.6}$$

where beta indicates the maximum gap increase, and lambda the threshold of independence signature membership beyond which the gap begins to fall. Figure 7.24

shows this as a closed-loop proportional control system, of the sort discussed in (Eyman 1988) with the set point (or "reference input") set to 1. The "scaling factor" λ was set to 0.01 since the mean independence membership always exceeds this value when the independence signature is present (see Figure 7.23). Since the choice of β was more problematic, several different values were tested. At the beginning of the experiment the I-value is practically zero because there is no idle gap between packets and hence no independence signature. However when I / λ becomes larger than 1, the input gap rapidly drops, but rises again when the independence signature disappears. Thus the input dispersion will oscillate at a value close to the point of onset of the independence signature. Figures 7.25 and 7.26 show typical results obtained for two values of β. For the smaller value it takes longer for the independence signature to be detected, but the amount oscillation around this value is much smaller. The following experiments are based upon the smaller β value, namely 0.0000005s.



7.24 Measurement mechanism as a proportional control system.

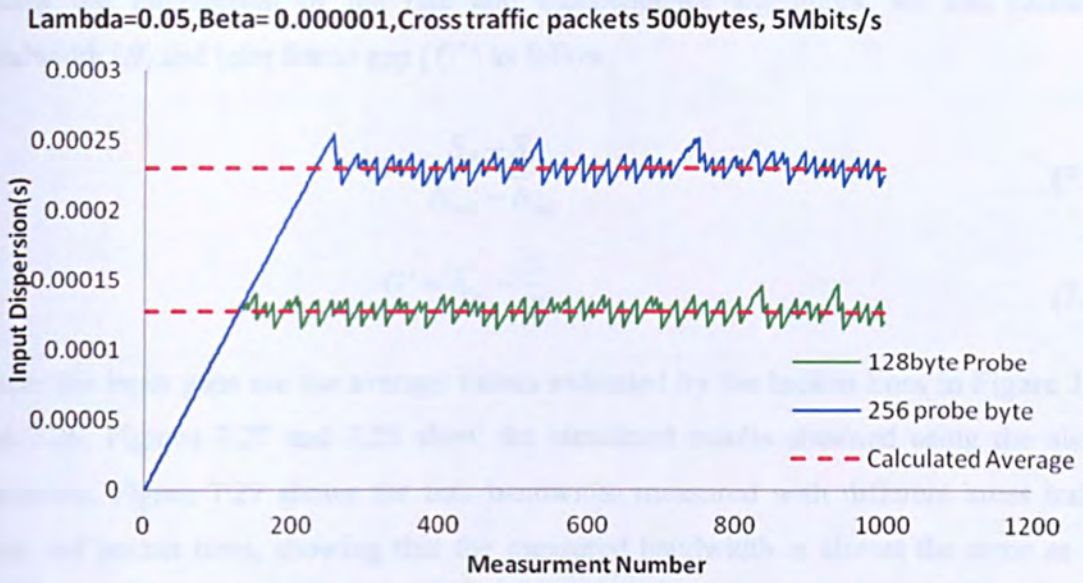
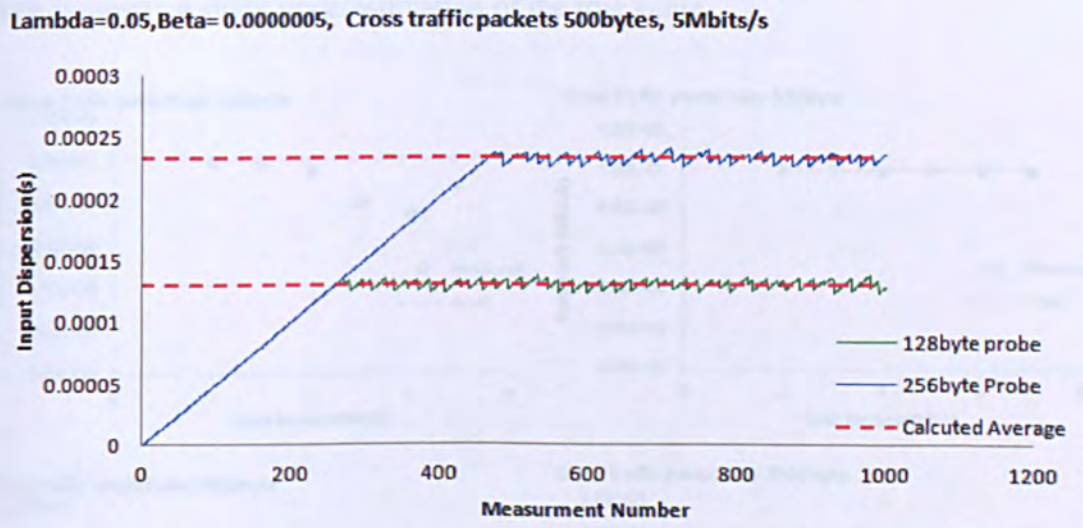


Figure 7.25 Controlled input gaps for the two packet sizes using beta=0.000001s.



7.26 Controlled input gaps for the two packet sizes using beta=0.0000005s.

7.6.1. Measuring the Link Parameters

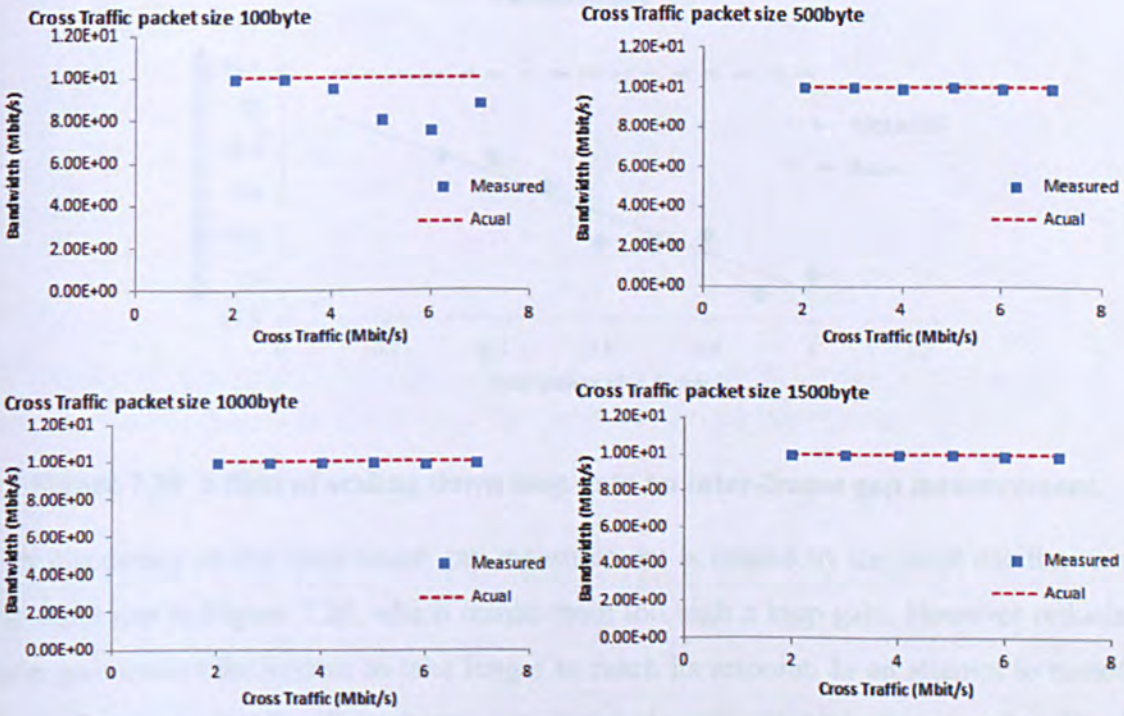
Figure 4.13 (in Chapter 4) shows that the rate signature output gap for packet size S_1 is $S_1/B + G'$ and hence the difference between the output gaps for two different packet sizes is $(S_1 - S_2)/B$. Since the input and output gaps detected by the above experiments

follow the intersection of the rate and independence signatures, we can calculate bandwidth (B) and inter frame gap (G') as follow.

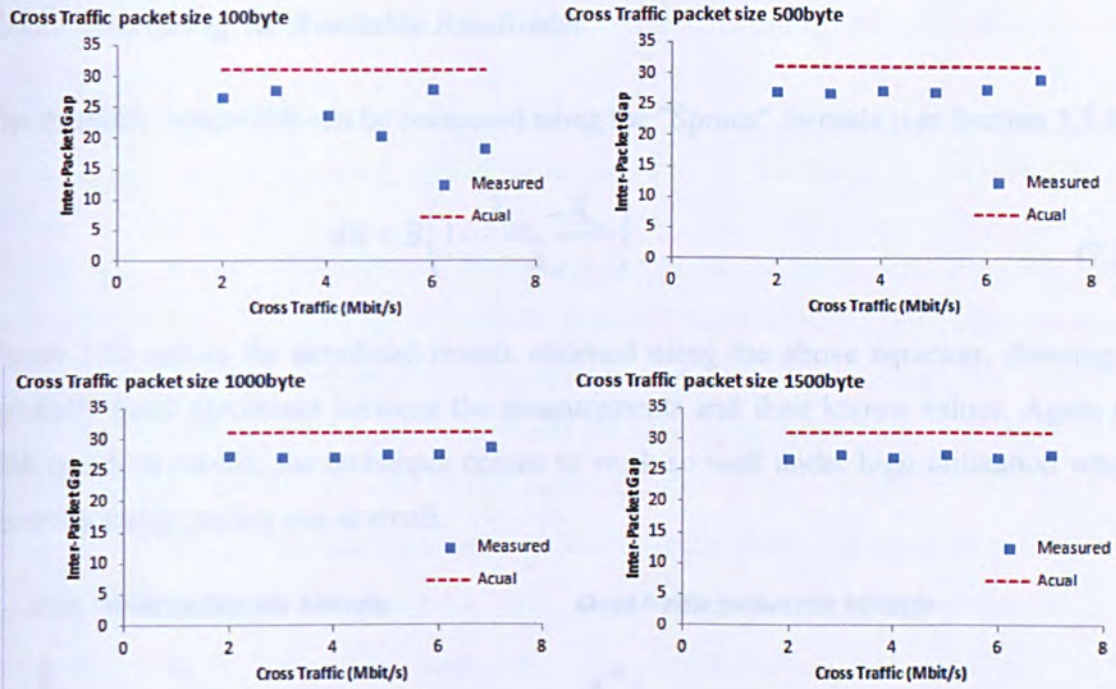
$$B = \frac{S_2 - S_1}{\overline{\Delta}_{in2} - \overline{\Delta}_{in1}} \tag{7.7}$$

$$G' = \overline{\Delta}_{in} - \frac{S}{B} \tag{7.8}$$

where the input gaps are the average values indicated by the broken lines in Figure 7.25 and 7.26. Figures 7.27 and 7.28 show the simulated results obtained using the above equations. Figure 7.27 shows the link bandwidth measured with different cross traffic rates and packet sizes, showing that the measured bandwidth is almost the same as the actual bandwidth with cross-traffic packet sizes 500, 1000 and 1500 bytes. However, with the smallest cross traffic packet size (100 bytes) problems occur when the utilization is high. This is probably because of the lack of a strong independence signature under such conditions (see Figures 7.16 and 7.17). Figure 7.28 shows the measured inter-packet gap, which is clearly a slight underestimation of the true value.



7.27 Link bandwidth measured from closed-loop probing data using different packet sizes and cross traffic rates compared with known values.



7.28 Inter-packet gap measured from closed-loop probing data using different packet sizes and cross traffic rates compared with known values.

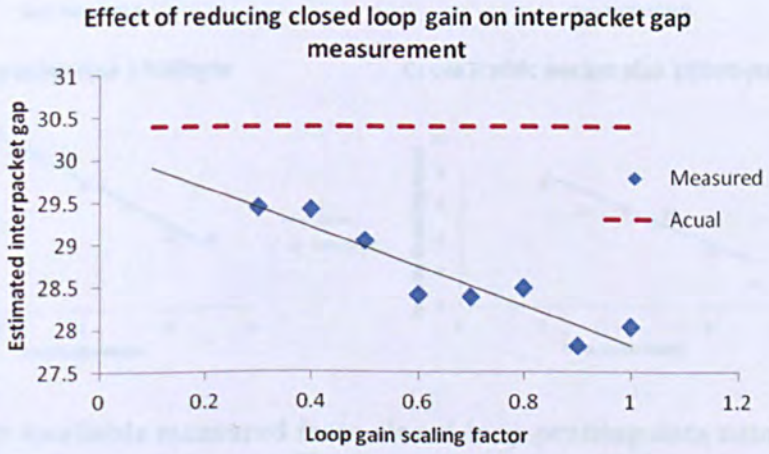


Figure 7.29 Effect of scaling down loop gain on inter-frame gap measurement.

The inaccuracy of the inter-frame gap measurement is caused by the large oscillation of the input gap in Figure 7.26, which results from too high a loop gain. However reducing loop gain causes the system to take longer to reach its setpoint. In an attempt to remedy this, a loop gain "scaling factor" was introduced after the setpoint was exceeded. Figure 7.29 shows how when this factor is gets smaller, the estimated inter-packet gap converges closer to its actual known value. However when the factor became too small (>0.3) the measurement technique failed and gave nonsensical results.

7.6.2. Measuring the Available Bandwidth

The available bandwidth can be computed using the "Spruce" formula (see Section 3.5.1):

$$AB = B.\left(1 - \frac{\overline{\Delta}_{out} - \overline{\Delta}_{in}}{\overline{\Delta}_{in}}\right). \tag{7.7}$$

Figure 7.30 shows the simulated results obtained using the above equation, showing a generally good agreement between the measurements and their known values. Again as with previous results, the technique ceases to work so well under high utilization when the cross traffic packet size is small.

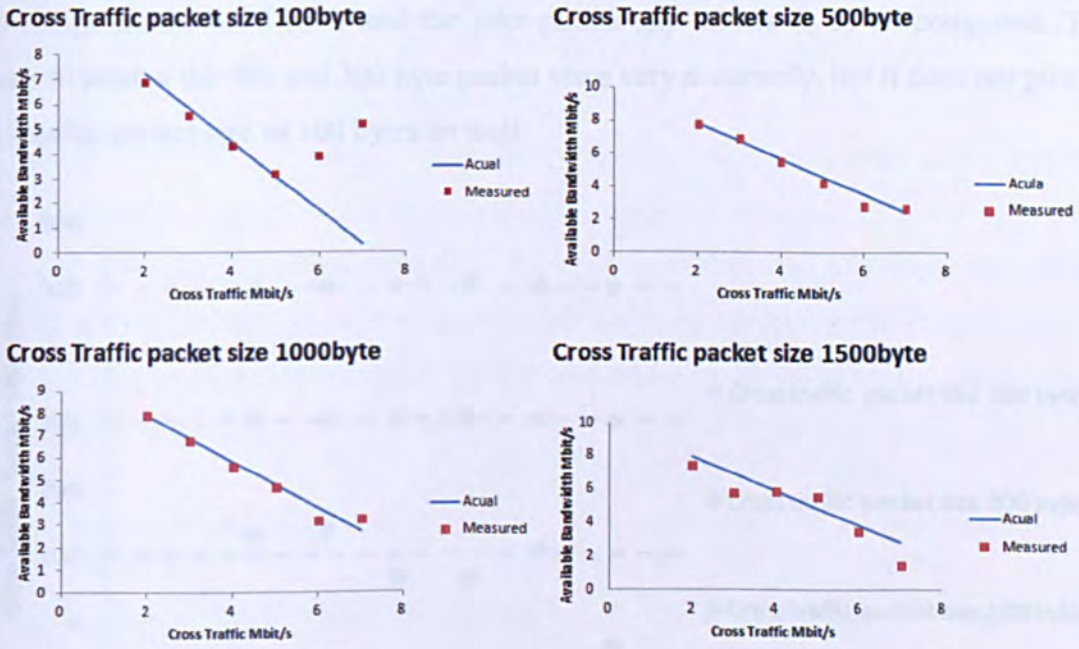


Figure 7.30 Available measured from closed-loop probing data using different packet sizes and cross traffic rates, compared with known values

In an attempt to solve the problem of smaller packet sizes under heavy cross-traffic, a further change to the algorithm was tried; when the weight of the independence signature in the long gaps fell below a certain level, the measured value of σ_{is} was deemed to be unreliable and a default value of 10 nanoseconds was automatically substituted. However, this tended not only to increase the under-prediction of the inter-packet gap, but also to

spoil the accuracy of the link and available bandwidth measurements. This solution was therefore rejected.

7.6.3. Measuring the Cross-Traffic Composition

Figure 7.31 shows the results of measuring the cross traffic packet size, using the Stauffer-Grimson algorithm (see Chapter 6) to model the dispersion distribution observed by the 128 byte probe packets. This was done by selecting the two weightiest modes in the model, which were assumed to represent the rate signature and the first distribution signature, since for small gaps the higher signatures should not be visible (see Chapter 5). Since the difference between these modes should theoretically be equal to $S_c/B+G'$, knowledge of the bandwidth and the inter-packet gap allows S_c to be computed. This seems to predict the 500 and 300 byte packet sizes very accurately, but it does not pick up the smaller packet size of 100 bytes so well.

Figure 7.31: Cross-traffic packet sizes predicted from Stauffer-Grimson modes. (Broken lines indicate the actual values.)

In view of the large variation of the 100 byte results, statistical significance tests were applied. Ten experiments were performed using S_c=100 and 500 bytes at 5Mbit/s and the 95% confidence intervals were computed. For 100 byte cross traffic, the 128 byte probe packet results gave an interval 84.77 to 100.61 bytes, which just barely contains the true value. However the 256 byte probe results gave a 130.01 to 169.01 byte interval, whose

lower limit is significantly higher than the true value. (A hypothesis test was also performed on the two data sets, which gave a p-value of 2.867×10^{-7}, indicating a significant dependence of the predicted value upon probe packet size.)

For 500 byte cross traffic the same experiment gave 95% confidence interval of 501.86 to 503.07 bytes for the 128 byte probe packets, and the 256 byte probe packet results gave an interval of 501.81 to 503.08 bytes. Both indicate statistically significant over-predictions, but the amount of over-prediction is very small.

7.7. Obtaining a Better Measurement

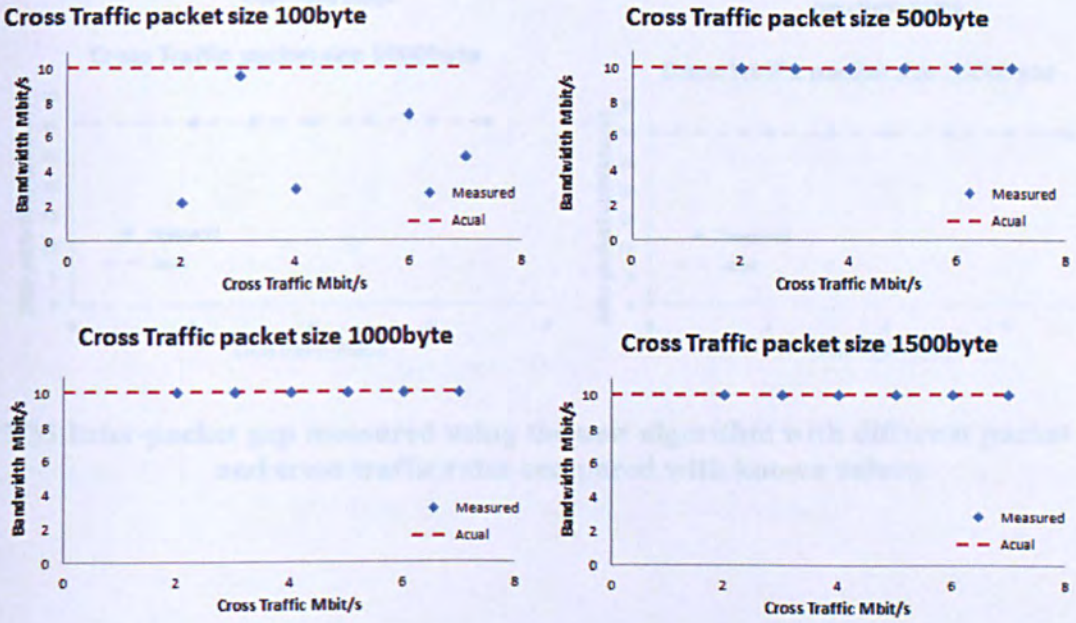
The previous section showed how a closed-loop control algorithm allowed the probing algorithm to "settle" at the onset of the independence signature, and the primary network/traffic parameters (link bandwidth, inter-packet gap, available bandwidth and cross-traffic packet size) to be determined. However multiple packet sizes introduce multiple distribution signatures, and the current algorithm assumes only a single packet size. This will normally be interpreted as the smallest packet size, and larger size packets will be ignored by the algorithm. In this section we propose an alternative approach which gives superior measurement results, and can potentially detect multiple packet sizes in the cross-traffic.

7.7.1. The Modified Algorithm

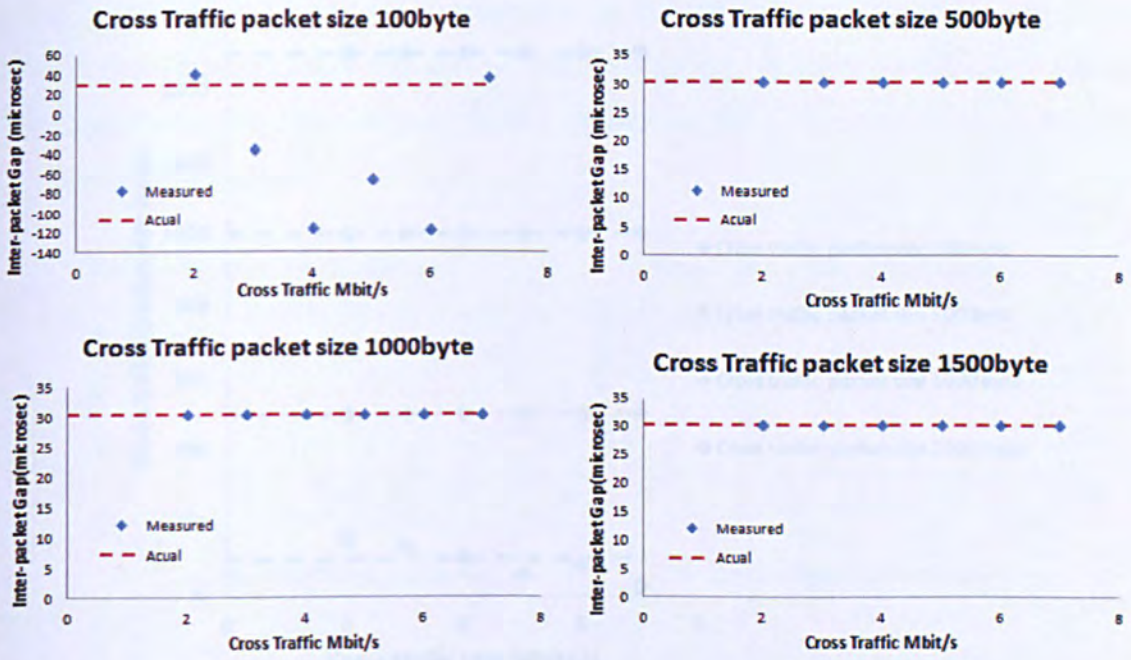
It is noticeable that when the input gap is small, the largest dispersion mode usually corresponds to the rate signature, so this can potentially be identified without the proportional control algorithm discussed above. For this reason, the new algorithm keeps input dispersion to a minimum (in this case zero) and collects data to identity rate signature. When 100 data points have been classified under the strongest mode, the input dispersion is raised to the mean value of this mode, and is held at this level for the remainder of the experiment. Link and traffic parameters are then extracted in same manner as before.

7.7.2. Results

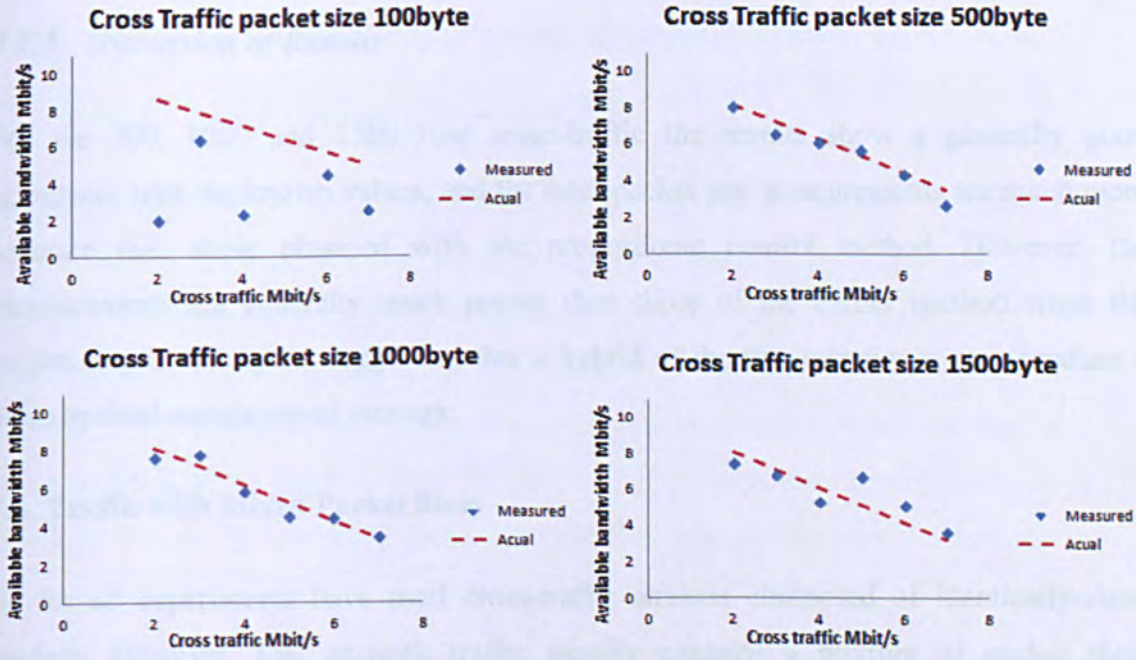
Figures 7.32, 7.33, 7.34 show the measured values of link bandwidth, inter-packet delay and available bandwidth. Figure 7.35 shows the results for different cross-traffic packet sizes (100,500,1000,1500 bytes).



7.32 Link bandwidth measured using the new algorithm with different packet sizes and cross traffic rates compared with known values.



7.33 Inter-packet gap measured using the new algorithm with different packet sizes and cross traffic rates compared with known values.



7.34 Available bandwidth measured using the new algorithm with different packet sizes and cross traffic rates compared with known values.

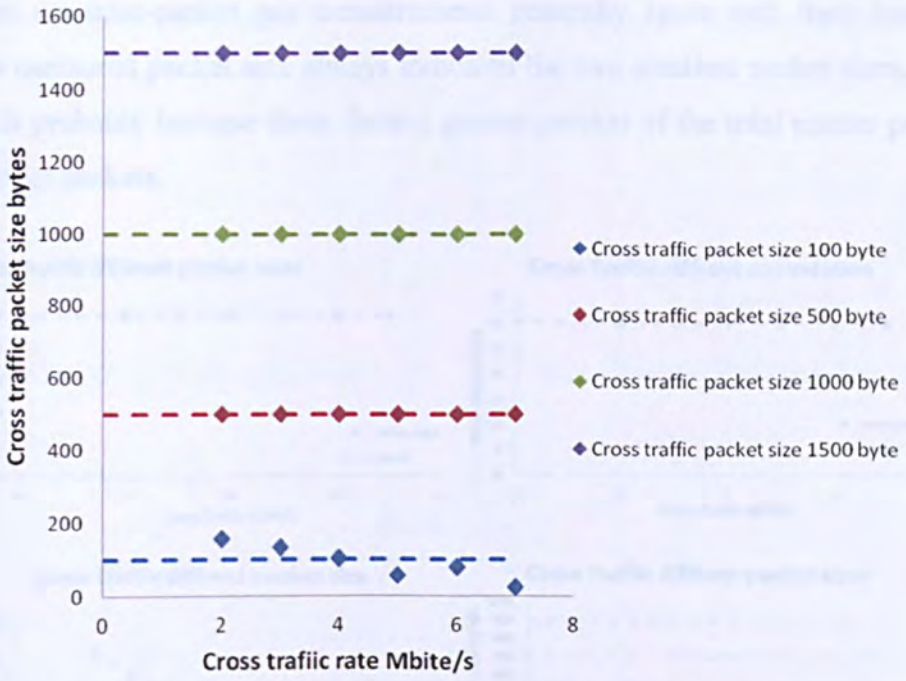


Figure 7.35 Cross-traffic packet sizes predicted from Stauffer-Grimson modes. (Broken lines indicate the actual values.)

7.7.3. Discussion of Results

For the 500, 1000 and 1500 byte cross-traffic the results show a generally good agreement with the known values, and the inter-packet gap measurements are much more accurate than those obtained with the proportional control method. However, the measurements are generally much poorer than those of the earlier method when the packet size is 100 bytes, suggesting that a hybrid of the two techniques may produce a more optimal measurement strategy.

7.8. Traffic with Mixed Packet Sizes

So far all experiments have used cross-traffic streams composed of identically-sized packets. However, real network traffic usually contains a mixture of packet sizes belonging to different kinds of networked application. The experiment was repeated with a mixture of packet sizes, 300, 500, 700 and 900 bytes (spanning most of the allowed packet-size range for Ethernet) each making up one quarter of the traffic load. Figure 7.36 shows the results. The measurements are generally good agreement with the known

values, and the inter-packet gap measurements generally agree with their true values, though the measured packet size always indicated the two smallest packet sizes, 300 and 500. This is probably because these form a greater portion of the total packet population than the larger packets.

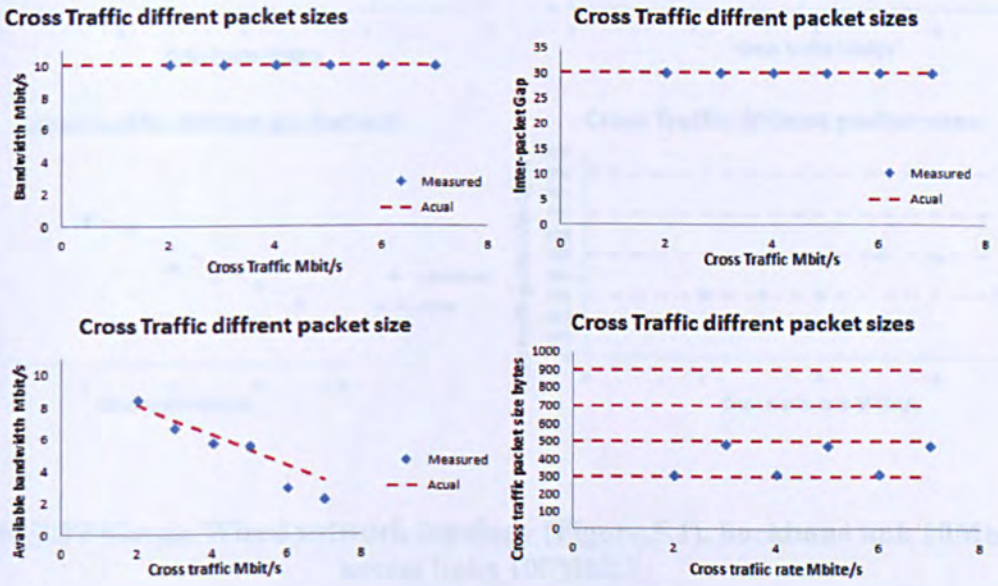


Figure 7.36 Results obtained using mixed packet sizes in cross traffic.

7.9. Networks with Dissimilar Link Capacities

So far, all the wired links in the network are assumed to be 10baseT, and therefore to have 10Mbit/s link rate. There is therefore no well-defined "narrow link" in the network, and the bottleneck is always the "tight-link" where the cross-traffic shares bandwidth with the probed path. The previous experiment was repeated with the end-links set to 100BaseTX (100Mbit/s Fast Ethernet) so the backbone link between the two switches became the physical bottleneck of the path (with 1Mbit/s link capacity) as well as the tight link. Figure 7.37 shows results almost identical to those previously reported.

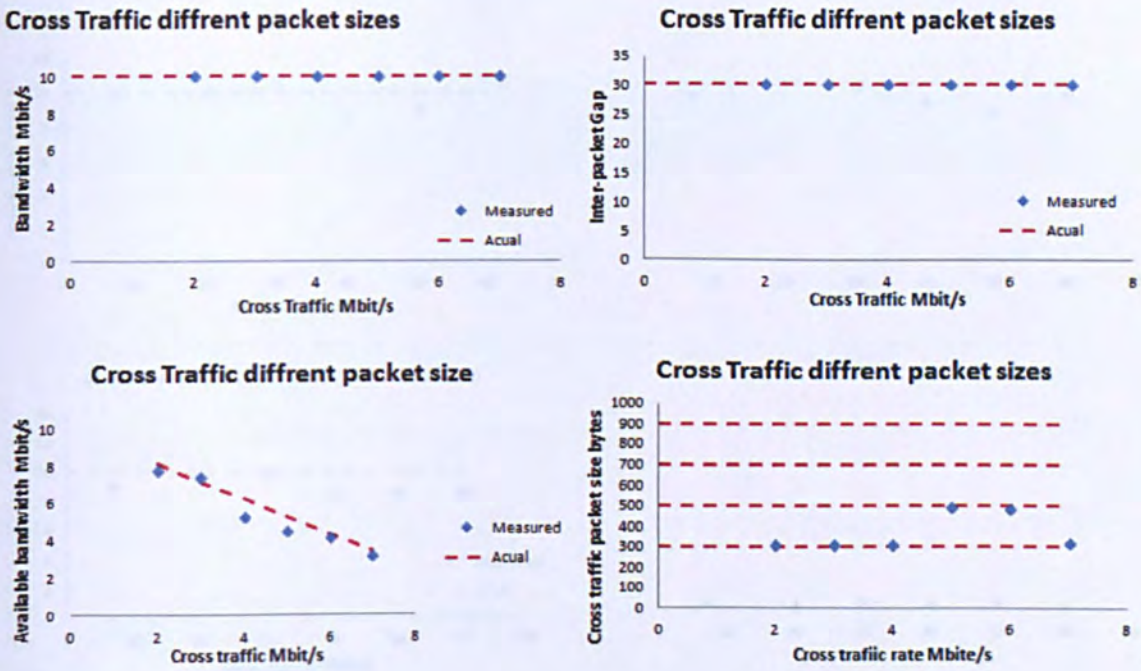


Figure 7.37 Simple Wired network topology (Figure.5.1), backbone link 10Mbit/s, access links 100Mbit/s

To investigate the opposite scenario the experiment has been performed again with the backbone fast Ethernet (100BaseTX) and access links 10BaseT. The latter are now the narrow links (10Mbit/s) while the central backbone link has 100Mbit/s. Figure 7.38 show the results using cross-traffic rates between 10 and 60Mbit/s. The measured bandwidth now points to the 10Mbit/s access links, though measurements are more variable than in earlier experiments due to disturbances from activity in the network core. The measured available bandwidth also points to this same value (10Mbit/s). The cross traffic in the core link does not significantly affect the quality of the connection, and therefore the packet size information is no longer reliable. However it is no longer so relevant since the bottleneck is now at the end links.

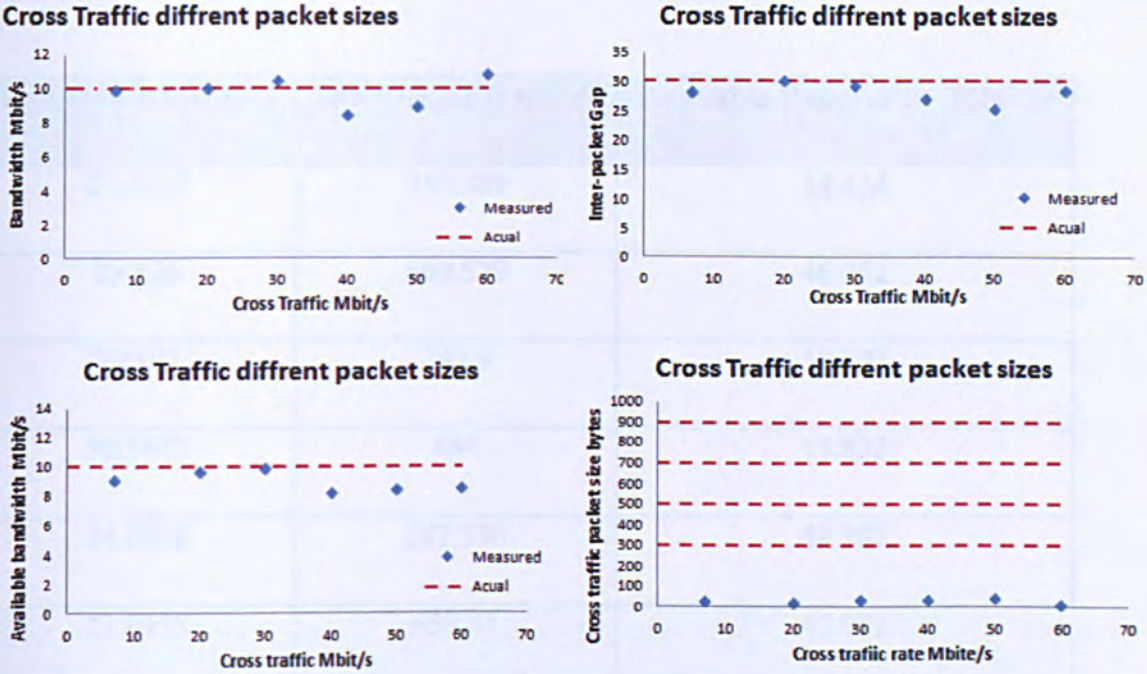


Figure 7.38 Simple Wired network topology (Figure.5.1), access links 10Mbit/s, backbone link 100Mbit/s

7.10. Application to Wired-cum-Wireless Networks

So far this chapter has dealt exclusively with wired networks, but an aim of this thesis was to investigate wired-cum-wireless networks also. In this section the close loop method (linear control algorithm, see section 7.6) has been applied to the wired-cum-wireless networks, together with the "improved" algorithm (see section 7.7) using both the first mile and last mile scenarios. Tables 7.2 and 7.3 show the link bandwidth, inter-frame gap and available bandwidth measured for repeated experiments using both the closed loop method and the improved algorithm for the first mile topology and Tables 7.4 and 7.5 show the corresponding results for last mile scenarios. The means and standard deviations for each data set are also shown.

Bandwidth (Mbit/s)	Inter frame Gap(μs)	Available Bandwidth (Mbit/s)
27.2167	350.389	54.434
23.126	669.579	46.252
29.103	382.9	58.207
59.1622	684	11.832
24.8938	287.336	49.787
23.9955	958.81	47.991
42.0233	753.681	84.0466
97.3083	614.302	19.461
18.48836	940.242	36.972
21.4268	953.419	42.852
Mean	**Mean**	**Mean**
36.6744	659.4658	45.18346
Standard Deviation	**Standard Deviation**	**Standard Deviation**
24.50368	253.1949	20.11791

Table 7.2 First Mile Results using the Closed Loop Method

Bandwidth (Mbit/s)	Inter frame Gap(μs)	Available Bandwidth (Mbit/s)
14.91	1034.85	14.05
10.50	895.03	93.52
14.59	768.60	11.12
35.31	590.41	28.54
31.06	679.45	28.08
Mean	**Mean**	**Mean**
21.27	793.67	35.06
Standard Deviation	**Standard Deviation**	**Standard Deviation**
11.11	175.66	33.63

Table 7.3 First Mile Results using the Improved Method

Bandwidth (Mbit/s)	Inter frame Gap(μs)	Available Bandwidth (Mbit/s)
17.5299	757.56	35.0599
28.3803	389.407	56.6606
36.7267	509.794	73.4535
33.562	465.622	67.1244
94.5424	859.214	18.908
55.152	532.732	11.0305
20.49997	921.681	40.999
11.5069	739.57	23.0139
33.979	480.902	66.195
10.3279	783.716	20.6557
Mean	**Mean**	**Mean**
34.22071	644.0198	41.31005
Standard Deviation	**Standard Deviation**	**Standard Deviation**
25.10275	188.1497	23.03065

Table 7.4 Last Mile Results using the Closed Loop Method

Bandwidth (Mbit/s)	Inter frame Gap(μs)	Available Bandwidth (Mbit/s)
35.1056	634.278	29.412
11.7327	967.134	10.5577
55.8628	682.136	44.399
51.093	634.818	38.6195
27.1777	363.393	17.1689
Mean	**Mean**	**Mean**
36.19	656.35	28.03
Standard Deviation	**Standard Deviation**	**Standard Deviation**
17.96	214.49	14.19

Table 7.5 Last Mile Results using the Improved Method

Two observations can be made: The measured bandwidths are much larger than their true values, and they show a tremendous variability between experiments. It was therefore concluded that the techniques devised are not suitable for wired-cum-wireless topologies because the large variable back off time in the wireless links prevents the formation of the precise signature modes on which both techniques depend.

7.11. Summery

This Chapter has explained how a closed loop probing algorithm has been applied to measure the link parameters, by examining the signature modes in the dispersion distribution during measurement. It may be summarised as follows:

- The ProbeSim Simulation Software was used to model the network. This is a free download software which includes several C++ classes from which bespoke simulations can be designed.
- Designs were verified against Opnet to prove the equivalence of the two softwares.
- A technique was devised for modelling the Independence Signature and detecting it in the results during measurement.
- A closed-loop control algorithm was devised which:
    - Measuried the Link Parameters
    - Measured the Available Bandwidth
    - Measured the Cross-Traffic Composition
- An improved method was devised which overcame some of the difficulties experienced in the closed-loop algorithm.
- The techniques were applied to scenarios where the traffic had Mixed Packet Sizes, networks had dissimilar link capacities and wired-cum-wireless networks.

It has been shown that a closed-loop linear control algorithm can be used to maintain the input gap at an optimum level (the onset of the independence signature) at which the bottleneck link bandwidth and cross-traffic information can be determined. The technique has its limitations however, and has been shown to be generally unsuitable for networks containing wireless links.

CHAPTER 8: DISCUSSION AND CONCLUSIONS

This final chapter summarizes the entire thesis and presents its overall findings, showing how the conclusions were reached from the data, and gives suggestions for future study.

8.1 Summary and Findings

The thesis began with a brief summary of wired and wireless Ethernet and some of the major technological issues associated with them. This included the historical background of Ethernet from its origin as Alohanet, the Legacy Ethernet protocol with CSMA/CD, switched Ethernet protocols, both wired and fibre-based, wireless Ethernet including WiFi and WiMAX and comparisons with other technologies. The different bandwidth measurement techniques were then compared, with an emphasis on packet pair measurements in wired and wireless networks. These included intermediate device-dependent methods (including Variable Packet Size (VPS) probing and SNMP-Based Measurements), methods which rely on one-way latencies, and end-to-end dispersion-based methods which use the bottleneck spacing effect. Of the latter, the Packet Pair/Train Dispersion (PPTD) and Train of Packet Pairs (TOPP) methods were selected as being of particular interest. The main reason for this was that the end-to-end dispersion approaches are usable even when it is impossible to access to other nodes in the path, and when there is no mechanism for clock synchronisation between endpoints.

To investigate this further, an Opnet simulation testbed for packet pair/stream probing was developed. This was tested using "baseline" experiments involving single wired and wireless links, to demonstrate the basic validity of the packet-pair method for bandwidth measurement. This was then extended to more complicated scenarios with multiple-hops and cross traffic, using three basic network topologies; wired, "first mile" wired-cum-

PAGE NUMBERING AS ORIGINAL

CHAPTER 8: DISCUSSION AND CONCLUSIONS

This final chapter summarizes the entire thesis and presents its overall findings, showing how the conclusions were reached from the data, and gives suggestions for future study.

8.1 Summary and Findings

The thesis began with a brief summary of wired and wireless Ethernet and some of the major technological issues associated with them. This included the historical background of Ethernet from its origin as Alohanet, the Legacy Ethernet protocol with CSMA/CD, switched Ethernet protocols, both wired and fibre-based, wireless Ethernet including WiFi and WiMAX and comparisons with other technologies. The different bandwidth measurement techniques were then compared, with an emphasis on packet pair measurements in wired and wireless networks. These included intermediate device-dependent methods (including Variable Packet Size (VPS) probing and SNMP-Based Measurements), methods which rely on one-way latencies, and end-to-end dispersion-based methods which use the bottleneck spacing effect. Of the latter, the Packet Pair/Train Dispersion (PPTD) and Train of Packet Pairs (TOPP) methods were selected as being of particular interest. The main reason for this was that the end-to-end dispersion approaches are usable even when it is impossible to access to other nodes in the path, and when there is no mechanism for clock synchronisation between endpoints.

To investigate this further, an Opnet simulation testbed for packet pair/stream probing was developed. This was tested using "baseline" experiments involving single wired and wireless links, to demonstrate the basic validity of the packet-pair method for bandwidth measurement. This was then extended to more complicated scenarios with multiple-hops and cross traffic, using three basic network topologies; wired, "first mile" wired-cum-

wireless, and "last mile" wired-cum-wireless. The distribution of output dispersion for each fixed value of input dispersion revealed the presence of discrete modes, which were explainable in terms of the independence, rate and distribution signatures previously identified by (Pásztor and Veitch 2002).

Though the early experiments used a simple histogram method for constructing dispersion probability distributions, it was recognized that an algorithm to pick out distribution modes (data clusters) was needed such that they could be mapped automatically to particular dispersion signatures. The Kernel Density algorithm was briefly considered, but this does not "pick out" modes as software entities and the estimated distribution must be scanned to identify maxima. The E-M k-Means algorithm on the other hand *does* represent individual modes as software entities, but it requires the number of modes, their heights and widths to be specified beforehand and merely adjusts their position to optimally represent the data. Experiments showed that it is not always very successful in doing even this. The third option investigated was the Stauffer-Grimson algorithm, which also represents modes as software entities, and requires no exact foreknowledge of the mode heights and widths. It is also inherently dynamic, and adjusts to changes in data by means of reinforcement learning. Experiments showed that it quickly identifies the histogram modes accurately. However, as well as the major signature modes it also produces lots of minor modes representing the independence noise and noise from random back offs, which needed to be filtered. It was decided, based on the experience of extensive tests, that the Stauffer-Grimson algorithm is the best option.

In the final phase of the research, a closed loop probing algorithm was used to optimize the probe parameters to measure the link characteristics, by examining the signature modes in the dispersion distribution during measurement. It may be summarised as follows. The ProbeSim software was used to model the network, which uses several C++ classes to construct bespoke simulations. These simulations were firstly verified against Opnet to prove the equivalence of the two softwares. Following successful validation a technique was devised for modelling the Independence Signature and detecting it in the results during measurement. Since this involved the measurement of time intervals of 0.5 seconds accurate to 0.1ns, its practical implementation would require a 32-bit counter clocked with a very stable timing signal (accurate to within 20μs per day). A closed-loop

control algorithm was then applied to keep the input gap equal to the rate signature such that the link parameters (including the maximum and available bandwidth) and the cross-traffic could be optimumly measured. An improved method was also devised which overcame some of the difficulties experienced in the closed-loop algorithm. The techniques were applied to scenarios where the traffic had mixed packet sizes, networks had dissimilar link capacities and wired-cum-wireless networks.

8.2 Future Work

The contribution of this work is to compliment and extends several existing studies on network probing. The use of of a closed-loop control algorithm in probing is not new: for example the Pathload algorithm (Jain and Dovrolis 2002) requires knowledge of the measured one-way delay in order to set the input rate in order to detect available bandwidth. However, the use of the Stauffer-Grimson algorithm for mode detection is an entirely new development introduced in this thesis, as are the active probing algorithms based upon it.

Several lines of future research can be identified. Firstly there are problems with the identification of multiple packet-sizes: the measured packet size always indicated only two smallest packet sizes (300 and 500 in these experiments). This is probably because these form a greater portion of the total packet population than the larger packets: larger packets will only be detected in the infrequent event that one should arrive between probe packets *without* any smaller packets arriving also. Consequently the algorithm works well when there is only one packet size but in other cases it does not work so well.

One possible approach to this problem would be to make greater use of the concurrent large and small inter-packet gaps associated with the large and small probe packets, thus capturing varying numbers of cross-traffic packets of differing sizes. Comparison of the resulting dispersion profiles may allow the different packet sizes to be distinguished.

Secondly there are still major problems with the probing of wired-cum-wireless links. Measured bandwidths are generally much larger than their known true values, and they furthermore show a tremendous variability between experiments. It was therefore concluded that the techniques devised are probably unsuitable (at least in their current form) for wired-cum-wireless topologies. This is because the large variable back off time

in the wireless links prevents the formation of the precise signature modes on which both techniques depend.

Before this last problem can be reasonably addressed, a better understanding of the effect of variable-latency links on the signature model would need to be established, together with an equivalent to the Stauffer-Grimson algorithm suitable for classifying wireless-link features. This would require a much more extensive simulation study of wired-cum-wireless behaviour that was possible in the time constraints of this project.

REFERENCES

Alliance, W. F. (2003). "Wi-Fi Protected Access: Strong, standards-based, interoperable security for today's Wi-Fi networks." White paper, University of Cape Town.

Amamra, A. and K. M. Hou (2008). SLOT: a fast and accurate technique to estimate available bandwidth in wireless IEEE 802.11, IEEE.

Axis-Communications (2010). "Local area network and Ethernet", http://www.axis.com/products/video/about_networkvideo/ip_networks.htm (Retrieved 29 Feburary, 2012).

Balakrishnan, H., V. N. Padmanabhan, et al. (1997). "A comparison of mechanisms for improving TCP performance over wireless links." Networking, IEEE/ACM Transactions on **5**(6): 756-769.

Balk, A., D. Maggiorini, et al. (2003). "Adaptive MPEG-4 video streaming with bandwidth estimation." Quality of Service in Multiservice IP Networks: 525-538.

Biaz, S. and N. H. Vaidya (1999). Discriminating congestion losses from wireless losses using inter-arrival times at the receiver, IEEE.

Bing, B. (2008). Emerging technologies in wireless LANs: Theory, design, and deployment, Cambridge Univ Pr.

Bredel, M. and M. Fidler (2008). "A measurement study of bandwidth estimation in IEEE 802.11 g wireless LANs using the DCF." NETWORKING 2008 Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet: 314-325.

Brenner, P. (1997). A Technical Tutorial on the IEEE 802.11 Protocol, Breezecom.

Carter, R. L. and M. E. Crovella (1996). Dynamic server selection using bandwidth probing in wide-area networks, Boston University Computer Science Department.

Castellanos, C. U., D. L. Villa, et al. (2006). Comparison of available bandwidth estimation techniques in packet-switched mobile networks, 17th Annual IEEE Conference on Personal, Mobile and Indoor Commucications, 11-14 September, Helsinki, Finland, pp.1-5.

Castro, R., M. Coates, et al. (2004). "Network tomography: Recent developments." Statistical Science: 499-517.

CCNA (2010). "Legacy Ethernet - Using Hubs." Retrieved 29th Feburary, 2012, from http://ciscoccna24.blogspot.com/2010/09/legacy-ethernet-using-hubs.html.

Cheng, S. T., Y. H. Cheng, et al. (2000). "WTCP: an Efficient Transmission Control Protocol for Wired/Wireless Internetworking." PROCEEDINGS-NATIONAL SCIENCE COUNCIL REPUBLIC OF CHINA PART A PHYSICAL SCIENCE AND ENGINEERING **24**(3): 176-185.

Dempster, A. P., N. M. Laird, et al. (1977). "Maximum likelihood from incomplete data via the EM algorithm." Journal of the Royal Statistical Society. Series B (Methodological): 1-38.

Dubois, X., L. S. T. B. Sorensen-Oumer, et al. (2005). Performance of different TCP versions over UMTS common/dedicated channels, Citeseer.

Elwalid, A. I. and D. Mitra (1993). "Effective bandwidth of general Markovian traffic sources and admission control of high speed networks." IEEE/ACM Transactions on Networking (TON) **1**(3): 329-343.

Eyman, E. (1988). Modeling,Simulation and Controling. St. Paul, West Publishing company.

Fairhurst, G. (2001). "Internet Communications Engineering - A Tutorial." Retrieved 28 Feburary, 2012, from http://www.erg.abdn.ac.uk/~gorry/eg3561/intro-pages/isp.html.

Fairhurst, G. (2011). "Wide Area Networks (WANs)." Retrieved 17 April, 2012, from http://www.erg.abdn.ac.uk/~gorry/eg3561/intro-pages/wan.html.

Franx, G. (2002). "The transient M/D/c queueing system." Preprint, available at http://www. cs. vu. nl/~ franx.

Fu, Z., B. Greenstein, et al. (2002). Design and implementation of a TCP-friendly transport protocol for ad hoc wireless networks, IEEE.

Fujistu (2006). Ethernet Tutorial, http://www.fujitsu.com/downloads/TEL/fnc/pdfservices/ethernet-prerequisite.pdf, (Accessed 24 October 2012).

Glover, I. A. and Grant,P.M. (2004). Digital Communication. Harlow, Prentice Hall.

Gupta, S.K., Kumar, R, (2011) "Reducing to Fault Errors in Communication Challel Systems", Int. Journal of Advances in Engineering and Tech., 1(4), pp.160-7.

Gupta, D., D. Wu, et al. (2009). Experimental comparison of bandwidth estimation tools for wireless mesh networks, IEEE.

Ha´ga, P. t., K. n. Diriczi, et al. (2007). "Granular mode of packet pair separation in poisson traffic." Computer Networks **51**: 683–698.

Herald, E. (2003). "Data Communincation Standards and Protocols." Retrieved 28th Feburary 2012, from http://www.eeherald.com/section/design-guide/ieee802_3.html.

Hosseinpour, M. and M. Tunnicliffe (2007). TOPP probing of network links with large independent latencies.

Hosseinpour, M. and M. Tunnicliffe (2009). Real-Time Tracking of Packet-Pair Dispersion Nodes Using the Kernel-Density and Gaussian-Mixture Models, IEEE.

Hu, N. and P. Steenkiste (2003). "Evaluation and characterization of available bandwidth probing techniques." Selected Areas in Communications, IEEE Journal on **21**(6): 879-894.

Jain, M. and C. Dovrolis (2002). End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput, ACM.

Jain, M. and C. Dovrolis (2002). Pathload: A measurement tool for end-to-end available bandwidth, Citeseer.

Jain, M. and C. Dovrolis (2004). Ten fallacies and pitfalls on end-to-end available bandwidth estimation, ACM.

Joanna, S. H. (2007). "Performance Study of a QoS Scheduling Algorithm over Wireless Networks." Retrieved 17 Dec, 2007, from http://summerrain.com/research/Technical_Paper_JS.pdf.

Johnsson, A., B. Melander, et al. (2006). "Bandwidth measurement in wireless networks." Challenges in Ad Hoc Networking: 89-98.

Kliazovich, D. and F. Granelli (2006). "Cross-layer congestion control in ad hoc wireless networks." Ad Hoc Networks **4**(6): 687-708.

Lai, K. and M. Baker (1999). Proc. of IEEE INFOCOM' 99, New York, NY, USA, March, pp.905-14.

Lakshminarayanan, K., V. N. Padmanabhan, et al. (2004). Bandwidth estimation in broadband access networks, ACM.

Lee, H. K., V. Hall, et al. (2007). "Bandwidth estimation in wireless LANs for multimedia streaming services." Advances in Multimedia **2007**(1): 9-9.

Leung, K. K., T. E. Klein, et al. (2004). Methods to improve TCP throughput in wireless networks with high delay variability [3G network example], IEEE.

Li, M., M. Claypool, et al. (2008). Wbest: a bandwidth estimation tool for ieee 802.11 wireless networks, IEEE.

Liu, X., K. Ravindran, et al. (2004). Single-hop probing asymptotics in available bandwidth estimation: sample-path analysis, ACM.

Liu, X., K. Ravindran, et al. (2005). What signals do packet-pair dispersions carry?, IEEE.

Mascolo, S., C. Casetti, et al. (2000). "TCP Westwood: congestion control with faster recovery." Univ. California, Los Angeles, Tech. Rep. CSD TR **200017**.

Mascolo, S. and F. Vacirca (2005). Congestion control and sizing router buffers in the internet, IEEE.

Melander, B., M. Bjorkman, et al. (2000). A new end-to-end probing and analysis method for estimating bandwidth bottlenecks, IEEE.

Melander, B., M. Bjorkman, et al. (2002). Regression-based available bandwidth measurements.

Mitchell, T. M. (1997). Machine Learning, New York, McGraw-Hill.

OPNET Technologies, I. OPNET Modeler Documentation Set. **15.0**.

Pahalawatta, P., R. Berry, et al. (2007). "Content-aware resource allocation and packet scheduling for video transmission over wireless networks." Selected Areas in Communications, IEEE Journal on **25**(4): 749-759.

Panko, R. R. (2005). Business Data Networks and Telecommunications. USA, Pearson Prentice Hall.

Park, K. J., H. Lim, et al. (2006). "Stochastic analysis of packet-pair probing for network bandwidth estimation." Computer Networks **50**(12): 1901-1915.

Parvez, N. and L. Hossain (2004). Improving TCP performance in wired-wireless networks by using a novel adaptive bandwidth estimation mechanism, IEEE.

Pásztor, A. and D. Veitch (2002). "On the scope of end-to-end probing methods." Communications Letters, IEEE **6**(11): 509-511.

Paxson, V. (1999). "End-to-end internet packet dynamics." IEEE/ACM Transactions on Networking (TON) **7**(3): 277-292.

Pentikousis, K. (2000). "TCP in wired-cum-wireless environments." Communications Surveys & Tutorials, IEEE **3**(4): 2-14.

Prasad, R., C. Dovrolis, et al. (2003). "Bandwidth estimation: metrics, measurement techniques, and tools." Network, IEEE **17**(6): 27-35.

Ribeiro, V., R. Riedi, et al. (2003). pathchirp: Efficient available bandwidth estimation for network paths.

Sarr, C., C. Chaudet, et al. (2008). "Bandwidth estimation for IEEE 802.11-based ad hoc networks." Mobile Computing, IEEE Transactions on **7**(10): 1228-1241.

Shah, S. H., K. Chen, et al. (2003). Available bandwidth estimation in IEEE 802.11-based wireless networks.

Shriram, A., M. Murray, et al. (2005). "Comparison of public end-to-end bandwidth estimation tools on high-speed links." Passive and Active Network Measurement: 306-320.

Sinha, R., N. Venkitaraman, et al. (2002). "WTC: A Reliable Transport Protocol for Wireless Wide-Area Networks " **VOl 8**(Number2-3): 301-316.

Spurgeon, C. E. (2003). designing and Managing Local Area Networks,Ethernet, The Definitive Guide.

Spurgeon, c. E. (2003). designing and Managing Local Area Networks,Ethernet, The Definitive Guide.

Stalling, W. (2004). Data and Computer Communication. Singapore, Person and Prentice Hall.

Stauffer, C. and W. E. L. Grimson (2000). "Learning patterns of activity using real-time tracking." Pattern Analysis and Machine Intelligence, IEEE Transactions on **22**(8): 747-757.

Strauss, J., D. Katabi, et al. (2003). A measurement study of available bandwidth estimation tools, ACM.

Suthaharan, S. and S. Kumar (2008). Measuring Available Bandwidth: pathChirp's Chirp Train Structure Remodeled, IEEE.

TechFest (1999). "Ethernet Technical Summary." Retrieved 28 February 2012, from http://www.techfest.com/networking/lan/ethernet4.htm.

TechTarget (2005). "ZigBee." Retrieved 29 Feburary, 2012, from http://searchmobilecomputing.techtarget.com/definition/ZigBee.

TechTerms (2012). "Bluetooth." Retrieved 25 Feb 2012, 2012, from http://www.techterms.com/definition/bluetooth.

Tse,D, Viswanath,P (2005) "Fundamentals of Wireless Communication", Cambridge University Press, New York.

Tunnicliffe, M. J. (2010). The Measurement of Bandwidth: A Simulation Study. Modeling, Simulation and Optimization Tolerance and Optimal Control. S. Cakaj, In-Teh: 285-304.

Tunnicliffe, M. J. (2010). "The measurement of bandwidth: A simulation study." Modelling, Simulation and Optimization", ed. Vedran Kordic (in press), INTECH.

Tunnicliffe, M. J. and M. Winnett (2009). "An approximate stochastic analysis of the packet-pair probing technique for available bandwidth estimation." International Journal of Communication Systems **22**(6): 651-669.

APPENDIX 1: PROBESIM CODE LISTINGS

I: Main Simulation Script (Lastmile Wireless)

```
//PROBING EXPERIMENT

#include <iostream>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
#include <math.h>
using namespace std;

#include "node.h"
#include "network.h"
#include "traffic.h"
#include "user.h"
#include "probe.h"
#include "agent.h"
#include "connection.h"
#include "channel.h"
#include "simulation.h"
#include "constants.h"
#include "model.h"
#include "indep_model.h"

void main()
{
	srand((unsigned)time(NULL));

	void simulate_agent(double simulation_time,double disp1,double disp2);

	double input_gap_1=0;
	double input_gap_2=0;

	simulate_agent(2000,input_gap_1,input_gap_2);

	cout<<"Simulation complete"<<endl;getch();
}

void simulate_agent(double simulation_time,double ingap_1,double ingap_2)
{
	/*********************************/
	////Probe-Agent Simulation

	int packet_size_1=128;
	int packet_size_2=256;
	double gap_between_pairs=0.5;
	double cross_traffic_rate=5e6;

	intno_of_pairs=(int)(simulation_time/gap_between_pairs);
```

```
double speed,fixed_ifg,variable_ifg;
int header_size,i;



/*********************************/
////Downstream Model

//Manager Host
speed=11e6;
fixed_ifg=364e-6;
variable_ifg=620e-6;
header_size=297;
node *MANAGERHOST=new node(1000000,speed,fixed_ifg);
MANAGERHOST->set_headersize(header_size);
MANAGERHOST->uniform_var(variable_ifg);

//Wireless Bridge (downstream)
speed=10e6;
fixed_ifg=12*8/speed;
header_size=18+8;
node *BRDS=new node(1000000,speed,fixed_ifg);
BRDS->set_headersize(header_size);

//Ethernet Switch 0 (downstream)
speed=10e6;
fixed_ifg=12*8/speed;
header_size=18+8;
node *SW0DS=new node(1000000,speed,fixed_ifg);
SW0DS->set_headersize(header_size);

//Ethernet Switch 1 (downstream)
speed=10e6;
fixed_ifg=12*8/speed;
header_size=18+8;
node *SW1DS=new node(1000000,speed,fixed_ifg);
SW1DS->set_headersize(header_size);

//Cross Traffic Source (downstream)
speed=10e6;
fixed_ifg=12*8/speed;
header_size=18+8;
node *CTDS=new node(1000000,speed,fixed_ifg);
CTDS->set_headersize(header_size);

/*********************************/
////Upstream Model

//Agent Host
speed=10e6;
fixed_ifg=12*8/speed;
header_size=18+8;
node *AGENTHOST=new node(1000000,speed,fixed_ifg);
AGENTHOST->set_headersize(header_size);

//Ethernet Switch 1 (upstream)
```

```
speed=10e6;
fixed_ifg=12*8/speed;
header_size=18+8;
node *SW1US=new node(1000000,speed,fixed_ifg);
SW1US->set_headersize(header_size);



//Ethernet Switch 0 (upstream)
speed=10e6;
fixed_ifg=12*8/speed;
header_size=18+8;
node *SW0US=new node(1000000,speed,fixed_ifg);
SW1US->set_headersize(header_size);

//Wireless Bridge (upstream)
speed=11e6;
fixed_ifg=364e-6;
variable_ifg=620e-6;
header_size=297;
node *BRUS=new node(1000000,speed,fixed_ifg);
BRUS->set_headersize(header_size);
BRUS->uniform_var(variable_ifg);

//Cross Traffic Source (upstream)
speed=10e6;
fixed_ifg=12*8/speed;
header_size=18+8;
node *CTUS=new node(1000000,speed,fixed_ifg);
CTUS->set_headersize(header_size);

/*********************************/
////Network Object

network *net=new network();

net->insert_node(MANAGERHOST);
net->insert_node(SW0DS);
net->insert_node(SW1DS);
net->insert_node(CTDS);

net->insert_node(AGENTHOST);
net->insert_node(SW1US);
net->insert_node(SW0US);
net->insert_node(CTUS);

net->insert_node(BRDS);
net->insert_node(BRUS);

int DS_route[]={0,1,2,8,-1};
int US_route[]={4,9,5,6,-1};
int CTDS_route[]={3,1,-1};
int CTUS_route[]={7,5,-1};

net->insert_route(DS_route);
net->insert_route(US_route);
net->insert_route(CTDS_route);
net->insert_route(CTUS_route);
```

```
/*********************************/
////Probe

//Establish probe process
Probe *p=new
probe(gap_between_pairs,2,ingap_1,ingap_2,packet_size_1,packet_size_2,
"C:\\ingap1.txt","C:\\.txt","C:\\ outgap1.txt","C:\\ outgap2.txt",
"C:\\longgap_diff.txt",false);

//Associate models with probe
model *model1=new model(),*model2=new model();
model *model1a=new model(),*model2a=new model();
p->add_models(model1,model2,model1a,model2a);

//Set for closed-loop operation
p->set_LOOP();

//Establish probe connection
connection *probeconnect=new connection();

//Associate probe process with connection
probeconnect->insert_probe(p);

//Associate connection with channel
channel *chp=new channel();
chp->insert_connection(probeconnect);

/*********************************/
////Agent

//Establish agent process
agent *a=new agent(100,1e-3,false);

//Associate agent with probe manager
a->set_probe(p);

//Establish agent connection
connection *agentconnect=new connection();

//Associate probe process with connection
agentconnect->insert_agent(a);

//Associate connection with channel
channel *cha=new channel();
cha->insert_connection(agentconnect);

/*********************************/
////Construct Cross Traffic

//Set granularity
int Sc[]={500,500,500,500};
```

```
double alpha[]={0.0477,0.0218,0.0950,0.8292};

//Establish cross-traffic processes
user *crosstraffic[8];
//Downstream
crosstraffic[0]=new user(Sc[0],0,false);
crosstraffic[0]
->set_stream(alpha[0]*cross_traffic_rate,true,NULL,NULL,0);
crosstraffic[1]=new user(Sc[1],0,false);
crosstraffic[1]
->set_stream(alpha[1]*cross_traffic_rate,true,NULL,NULL,0);
crosstraffic[2]=new user(Sc[2],0,false);
crosstraffic[2]
->set_stream(alpha[2]*cross_traffic_rate,true,NULL,NULL,0);
crosstraffic[3]=new user(Sc[3],0,false);
crosstraffic[3]
->set_stream(alpha[3]*cross_traffic_rate,true,NULL,NULL,0);

//Upstream
crosstraffic[4]=new user(Sc[0],0,false);
crosstraffic[4]
->set_stream(alpha[0]*cross_traffic_rate,true,NULL,NULL,0);
crosstraffic[5]=new user(Sc[1],0,false);
crosstraffic[5]
->set_stream(alpha[1]*cross_traffic_rate,true,NULL,NULL,0);
crosstraffic[6]=new user(Sc[2],0,false);
crosstraffic[6]
->set_stream(alpha[2]*cross_traffic_rate,true,NULL,NULL,0);
crosstraffic[7]=new user(Sc[3],0,false);
crosstraffic[7]
->set_stream(alpha[3]*cross_traffic_rate,true,NULL,NULL,0);

//Establish cross-traffic connections
connection *crossconnect[8];
for (i=0;i<8;i++) crossconnect[i]=new connection();

//Associate cross-traffic processes with connections
for (i=0;i<8;i++) crossconnect[i]->insert_user(crosstraffic[i]);

//Associate connections with channels
channel *ch[2];

ch[0]=new channel();
ch[0]->insert_connection(crossconnect[0]);
ch[0]->insert_connection(crossconnect[1]);
ch[0]->insert_connection(crossconnect[2]);
ch[0]->insert_connection(crossconnect[3]);

ch[1]=new channel();
ch[1]->insert_connection(crossconnect[4]);
ch[1]->insert_connection(crossconnect[5]);
ch[1]->insert_connection(crossconnect[6]);
ch[1]->insert_connection(crossconnect[7]);

/*********************************/
////Construct Traffic Object
```

```
        traffic *traff=new traffic();

        traff->insert_channel(chp);
        traff->insert_channel(cha);

        for (i=0;i<2;i++) traff->insert_channel(ch[i]);

        /*********************************/
        ////Simulation Object

        simulation *sim=new simulation();
        sim->configure(net,traff);
        sim->add_jitter(1e-9);

        /*********************************/
        ////Run Simulation

        p->openfiles();
        double end_time=sim->advance(simulation_time);
        p->closefiles();

        cout<<"The Bandwidth is "<<p->get_bandwidth()<<endl;
        cout<<"The Inter-Packet Gap is "<<p->get_gap()<<endl;
        cout<<"The AB is "<<p->get_AB()<<endl;
        cout<<"The packet size is "<<p->get_packetsize(1)<<"  "
                                        <<p->get_packetsize(2)<<endl;
}
```

Probe Receive Process

(a) Initial Version

```
#define BETA 0.00001
#define LAMBDA 0.05

void    probe::receive_agent_data(double    long_indisp,double    long_outdisp,double
short_indisp,double short_outdisp,int parstream)
{
        //Receive input/output dispersion from an agent packet

        //Save data to files
        if (parstream==1)
        {
                ingapfile_1<<short_indisp<<endl;
                outgapfile_1<<short_outdisp<<endl;
        }
        else
        {
                ingapfile_2<<short_indisp<<endl;
                outgapfile_2<<short_outdisp<<endl;
        }

        if (LOOP)
```

```
{
	//Closed-loop functionality

	//Adjust independence model
	model_i->add_data(long_outdisp-long_indisp);
	double I=model_i->independence(short_outdisp-short_indisp);

	if (parstream==1)
	{
		if (measure1)
		{
			//Measurement phase
			model1->add_data(short_outdisp);
			in_separation_1=model1->get_mean_largest();
		}
		else
		{
			if (I<LAMBDA)
			{
				//Initial rise phase
				model1->add_data(short_outdisp);
				in_separation_1+=BETA;
			}
			else
			{
				//Trigger measurement phase
				in_separation_1=model1->get_mean_largest();
				measure1=true;
			}
		}
	}
	else
	{
		if (measure2)
		{
			//Measurement phase
			model2->add_data(short_outdisp);
			in_separation_2=model2->get_mean_largest();
		}
		else
		{
			if (I<LAMBDA)
			{
				//Initial rise phase
				model2->add_data(short_outdisp);
				in_separation_2+=BETA;
			}
			else
			{
				//Trigger measurement phase
				in_separation_2=model2->get_mean_largest();
				measure2=true;
			}
		}
	}
}

if (measure1&&measure2)
{
	number++;

	//Determine bandwidth
```

```
                double thisbandwidth
                        =8*(double)(packetsize_2-packetsize_1)
                        /(in_separation_2-in_separation_1);
                counter+=thisbandwidth;
                bandwidth=counter/number;

                //Determine interframe gap
                double thisgap=in_separation_2-8*(double)packetsize_2/bandwidth;
                counter1+=thisgap;
                interpacket_gap=counter1/number;

                //Available bandwidth
                incount+=short_indisp;
                outcount+=short_outdisp;
                avail_bandwidth=bandwidth*(1-(outcount-incount)/incount);
        }
}
```

(b) Improved Version

```
#define RATE_HITS 100

void    probe::receive_agent_data(double    long_indisp,double    long_outdisp,double
short_indisp,double short_outdisp,int parstream)
{
        //Receive input/output dispersion from an agent packet

        //Add data to to files

        if (parstream==1)
        {
                ingapfile_1<<short_indisp<<endl;
                outgapfile_1<<short_outdisp<<endl;
        }
        else
        {
                ingapfile_2<<short_indisp<<endl;
                outgapfile_2<<short_outdisp<<endl;
        }

        //Update summary results
        if (LOOP)
        {
                //Closed-loop functionality
                int hits;
                double maxmean;

                if (parstream==1)
                {
                        //Packet size 1
                        if (!measure_1)
                        {
                                //Acquiring rate signature
                                model1->add_data(short_outdisp);
                                maxmean=model1->mean_heaviest(hits);
                                if (hits>=RATE_HITS)
                                {
```

```
                                        //Rate signature acquired
                                        measure_1=true;
                                        in_separation_1=maxmean;
                                }
                        }
                        else
                        {
                                //Rate signature already acquired
                                model1a->add_data(short_outdisp);
                                tot_out_sep_1+=short_outdisp;
                                num_sep_1++;
                        }
                }
                else
                {
                        //Packet size 2
                        if (!measure_2)
                        {
                                //Acquiring rate signature
                                model2->add_data(short_outdisp);
                                maxmean=model2->mean_heaviest(hits);
                                if (hits>=RATE_HITS)
                                {
                                        //Rate signature acquired
                                        measure_2=true;
                                        in_separation_2=maxmean;
                                }
                        }
                        else
                        {
                                //Rate signature acquired
                                model2a->add_data(short_outdisp);
                                tot_out_sep_2+=short_outdisp;
                                num_sep_2++;
                        }
                }
        }
}
```

Stauffer Grimson Model

```
#ifndef MODEL_H
#define MODEL_H

#include "process.h"
#include "population.h"

class model
{
private:
        population thispopulation;

public:
        model();
        void add_data(double);
        double get_pdf(double);
        void add_process(process*);
        bool save_model(char *meanfile,char *stdvfile, char *weightfile);
        double diff_2();
```

```
        double mean_heaviest(int &hits);
};

#endif

#include <math.h>
#include <conio.h>
#include <iostream>
#include <fstream>
using namespace std;

#include "model.h"

#define NEWSTDEV 0.00001
#define NEWWEIGHT 0.0001
#define HIDDEN_RATIO 0.1

model::model()
{
        //Create empty model
        thispopulation.reset();
}
void model::add_data(double x)
{
        //Add new data to model

        //Find the process best-fit to receive the data
        process *thisprocess,*maxprocess,*newprocess;
        double maxvalue=0.0;
        double thisvalue;
        if (thispopulation.getnumprocesses()>0)
        {
                do
                {
                        thisprocess=thispopulation.get_process();
                        thisvalue=thisprocess->pdf_value(x,true);
                        if (thisvalue>maxvalue)
                        {
                                maxprocess=thisprocess;
                                maxvalue=thisvalue;
                        }

                        //Depreciate mode significance
                        thisprocess->depreciate();
                }
                while (thispopulation.next_process());
                thispopulation.reset();
        }

        //Add data to successful process (if any)
        if (maxvalue>0.0) maxprocess->add_data(x);
        else
        {
                //Create new process to acquire new data-point
                newprocess=new process(x,NEWSTDEV,NEWWEIGHT);
                thispopulation.add_process(newprocess);
        }

        //Normalize model
        double total=0;
        do total+=thispopulation.get_process()->get_weight();
```

```
        while (thispopulation.next_process());
        thispopulation.reset();
        do
        {
                thisprocess=thispopulation.get_process();
                thisprocess->set_weight(thisprocess->get_weight()/total);
        }
        while (thispopulation.next_process());
        thispopulation.reset();
}

double model::get_pdf(double x)
{
        //Return value of the pdf
        double total=0;

        do total+=thispopulation.get_process()->pdf_value(x,false);
        while (thispopulation.next_process());
        thispopulation.reset();
        return total;
}

bool model::save_model(char *meanfilename,char *stdvfilename,char *weightfilename)
{
        //Save model to file
        if (thispopulation.getnumprocesses()==0) return false;
        if (!meanfilename) return false;
        if (!stdvfilename) return false;
        if (!weightfilename) return false;

        ofstream meanfile,stdvfile,weightfile;
        meanfile.open(meanfilename);
        stdvfile.open(stdvfilename);
        weightfile.open(weightfilename);

        process *thisprocess;
        do
        {
                thisprocess=thispopulation.get_process();
                meanfile<<thisprocess->get_mean()<<endl;
                stdvfile<<thisprocess->get_stdev()<<endl;
                weightfile<<thisprocess->get_weight()<<endl;
        }
        while (thispopulation.next_process());
        thispopulation.reset();

        meanfile.close();
        stdvfile.close();
        meanfile.clear();
        stdvfile.clear();
        weightfile.close();
        weightfile.clear();

        return true;
}

double model::diff_2()
{
        //Returns difference in mean between the heaviest and next heaviest mode
        process *thisprocess,*maxprocess;
        double meanval[2],maxweight;
        for (int i=0;i<2;i++)
```

```
	{
		maxweight=0;
		do
		{
			thisprocess=thispopulation.get_process();
			if (thisprocess->get_weight()
					>maxweight&&!thisprocess->ismarked())
			{
				maxprocess=thisprocess;
				maxweight=thisprocess->get_weight();
			}
		}
		while (thispopulation.next_process());
		thispopulation.reset();
		meanval[i]=maxprocess->get_mean();
		maxprocess->mark();
	}

	//Unmark processes
	do
	{
		thisprocess=thispopulation.get_process();
		thisprocess->unmark();
	}
	while (thispopulation.next_process());
	thispopulation.reset();

	return fabs(meanval[0]-meanval[1]);
}

double model::get_mean_largest()
{
	//Returns lowest mean of 2 largest Gaussian components
	process *thisprocess,*maxprocess;
	double meanval[2],maxweight;
	for (int i=0;i<2;i++)
	{
		maxweight=0;
		do
		{
			thisprocess=thispopulation.get_process();
			if (thisprocess->get_weight()
					>maxweight&&!thisprocess->ismarked())
			{
				maxprocess=thisprocess;
				maxweight=thisprocess->get_weight();
			}
		}
		while (thispopulation.next_process());
		thispopulation.reset();
		meanval[i]=maxprocess->get_mean();
		maxprocess->mark();
	}

	//Unmark processes
	do
	{
		thisprocess=thispopulation.get_process();
		thisprocess->unmark();
	}
	while (thispopulation.next_process());
	thispopulation.reset();
```

```
	if (meanval[0]>meanval[1]) return meanval[1];
	return meanval[0];
}

#ifndef POPULATION_H
#define POPULATION_H

#include "process.h"

class population
{
private:
	process *first,*last,*current;
	int no_of_processes;
	bool got_to_end;

public:
	population();
	void add_process(process*);
	process *get_process();
	int getnumprocesses(){return no_of_processes;}
	bool next_process();
	void reset();
	bool reached_end(){return got_to_end;}
};

#endif

#include <math.h>
#include <conio.h>
#include <iostream>
#include <fstream>
using namespace std;

#include "population.h"

population::population()
{
	//Create empty model
	no_of_processes=0;
	first=NULL;
	last=NULL;
	current=NULL;
	got_to_end=false;
}

void population::add_process(process *newprocess)
{
	//Add a new process to the model
	if (no_of_processes==0)
	{
		first=newprocess;
		last=newprocess;
		current=newprocess;
	}
	else
	{
		last->set_next(newprocess);
		newprocess->set_previous(last);
		last=newprocess;
```

```
        }
        no_of_processes++;
}

process* population::get_process()
{
        //Return current process
        return current;
}

bool population::next_process()
{
        //Move current to next process
        if (current!=last)
        {
                current=current->get_next();
                return true;
        }
        got_to_end=true;
        return false;
}

void population::reset()
{
        //Set current process to first
        current=first;
        got_to_end=false;
}

#ifndef PROCESS_H
#define PROCESS_H

class process
{
private:
        double mean;
        double stdev;
        double weight;
        bool marked;

        process *previous,*next;

public:
        process(double mean,double stdev,double weight);
        void add_data(double x);
        void depreciate();
        double pdf_value(double x,bool clip);
        process *get_next(){return next;}
        void set_next(process *n){next=n;}
        process *get_previous(){return previous;}
        void set_previous(process *p){previous=p;}
        double get_mean(){return mean;}
        double get_stdev(){return stdev;}
        double get_weight(){return weight;}
        void set_weight(double w){weight=w;};
        process *clone();
        void mark(),unmark();
        bool ismarked();
};

#endif
```

```
#include <math.h>
#include <stdlib.h>
#include <iostream>
#include <conio.h>
using namespace std;

#include "process.h"

#define PI 3.141592654
#define ALPHA 0.01

process::process(double m,double s,double w)
{
	//Create new process
	mean=m;
	stdev=s;
	weight=w;
	previous=NULL;
	next=NULL;

	marked=false;
}

void process::add_data(double x)
{
	//Appreciate significance
	weight+=ALPHA;

	//Adjust mean and standard deviation
	double rho=ALPHA*exp(-pow((x-mean)/stdev,2)/2);
	mean=(1-rho)*mean+rho*x;
	stdev=sqrt((1.0-rho)*pow(stdev,2)+rho*pow(x-mean,2));
}

void process::depreciate()
{
	//Depreciate significance
	weight=(1.0-ALPHA)*weight;
}

double process::pdf_value(double x,bool clip)
{Re
	//turn value of the process' PDF
	if (fabs(x-mean)>2.5*stdev&&clip) return 0.0;
	return weight/(stdev*sqrt(2.0*PI))*exp(-pow((x-mean)/stdev,2)/2);
}

process *process::clone()
{
	process *cloneprocess=new process(mean,stdev,weight);
	return cloneprocess;
}

void process::mark()
{
	//Mark process
	marked=true;
}

void process::unmark()
{
	//Unmark process
```

```
	marked=false;
}

bool process::ismarked()
{
	//Returns true if process is marked
	return marked;
}
```

Independence Signature Model

```
#ifndef INDEP_MODEL_H
#define INDEP_MODEL_H

class indep_model
{
private:
	double weight1,weight0;
	double sigma;
	double alpha;

public:
	indep_model();
	void add_data(double x);
	double get_weight();
	double get_stdev();
	double independence(double x);
};

#endif

#include <fstream>
#include <iostream>
#include <conio.h>
using namespace std;

#define ALPHA 0.01
#define NEWSTDEV 1e-9
#define NEWWEIGHT 0.00001

#include "indep_model.h"

indep_model::indep_model()
{
	weight1=NEWWEIGHT;
	weight0=1.0-weight1;
	sigma=NEWSTDEV;
	alpha=ALPHA;
}

void indep_model::add_data(double x)
{
	//Add new data to model
	if (fabs(x)<2.5*sigma)
	{
		//new datum classified as independent

		//Adjust weights
		weight1=(1.0-alpha)*weight1+alpha;
```

```
                weight0=(1.0-alpha)*weight0;

                //Adjust standard deviation
                sigma=sqrt((1.0-alpha)*pow(sigma,2)+alpha*pow(x,2));
        }
        else
        {
                //New datum classified as non-independent
                weight0=(1.0-alpha)*weight0+alpha;
                weight1=(1.0-alpha)*weight1;
        }

        //Normalize weights
        double totweight=weight0+weight1;
        weight0/=totweight;
        weight1/=totweight;
}

double indep_model::get_weight()
{
        //Return independence weight
        return weight1;
}

double indep_model::get_stdev()
{
        //Return standard deviation
        return sigma;
}

double indep_model::independence(double x)
{
        //Return fuzzy set-membership of x to independence signature
        return exp(-pow(x/sigma,2)/2);
}
```

PAGE/PAGES EXCLUDED UNDER INSTRUCTION FROM THE UNIVERSITY

APPENDIX3 : Historical Background

A3.1 Early Development

The development of Ethernet can be traced back to the University of Hawaii in the late 1960's. Here researchers developed a system called Aloha Net, for communicating wirelessly between different islands. The protocol used under Aloha was really simple: each transmitting station could send a packet whenever it needed to, and then waited for an acknowledgment from the receiving station. If the acknowledgement did not arrive in a short period of time, the system assumed that the message had "collided" with another transmission, and that both transmissions had become garbled. The transmitting stations would resend the packets when collision was detected; the success rate of delivery was higher after detecting collision. (Spurgeon 2003)

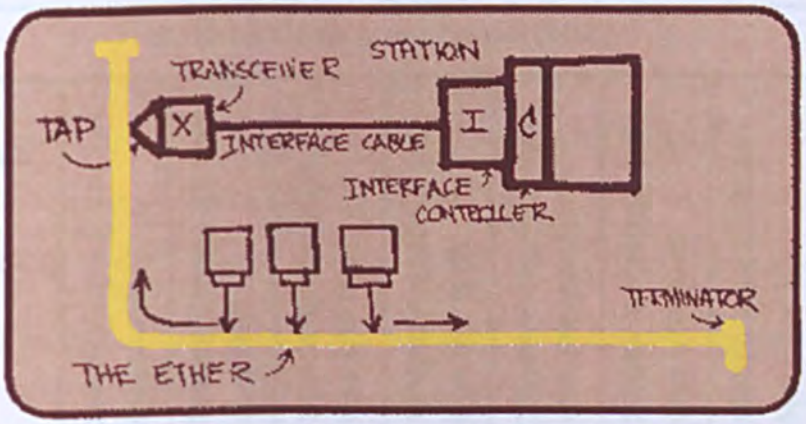


Figure A3.0.1 An early conceptual Ethernet sketch (Spurgeon 2003).

In the early 1970s, Harvard graduate student Robert Metcalfe (who in 1981 founded 3Com), working at the Xerox Plato Alto Research Centre, learned of Aloha Net and reasoned that some mathematical enhancements would increase its efficiency. The wired network protocol he designed came to be called Ethernet, in which the elegance and the simplicity of the Aloha Net were maintained. Figure 2.1 shows an early sketch of an Ethernet network: The "Ether" is a wire bus to which all transmitting and receiving

stations are connected, and thus forms a common medium of communication. Ethernet data units (or "frames"), which have a source and destination address followed by the packet payload and error checking data, occupy the bus during transmission (half duplex), and simultaneous transmissions result in collision. When a collision is detected, all stations cease transmission and back off for a random length of time. This system is called Carrier Sense Multiple Access with Collision Detection (CSMA/CD).

The addresses (known as the Media Access Control or MAC addresses) are hard-wired into the Ethernet interface cards, so manufacturers can use only the addresses in their assigned series, and use each one only once: That way, no two computers in the world can have the same address.

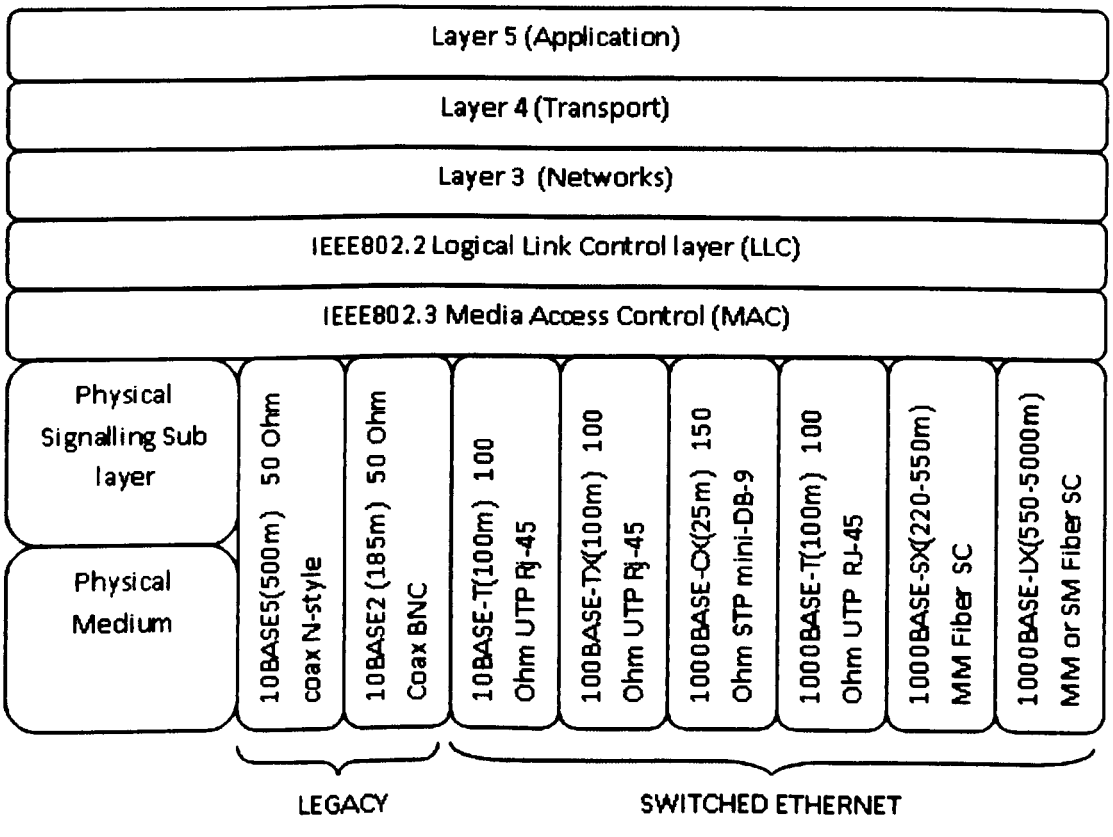


Figure A3.2 IEEE 802 Ethernet Standards, shown together with upper TCP/IP layers.

A3.1.2 Ethernet Development under the IEEE

Because of the success of Ethernet, in 1980 the Institute of Electrical and Electronic Engineers (IEEE) took it over as Working Group 802. Further evolution took place under IEEE, which also set standards for copper, fibre and wireless transmission. However, Ethernet fails to provide a complete solution to the problem, so must be combined with an internet protocol, and the synthesis of TCP/IP and Ethernet has solved many problems.

CSMA/CD is an example of "baseband" transmission, in which binary signals are placed directly on transmission medium without modulation. The alternative is "broadband" in which the binary information modulates a carrier signal. Baseband is much cheaper than the broadband, and is used when signals are transmitted across small distances. The entire bandwidth is used to transmit just one signal, unlike broadband in which multiple pieces of data are transferred together to increase the effectiveness of the transmission.

The first Ethernet standard specified a bandwidth of 2.94 Mbps (roughly 300,000 characters per second) while the more advanced IEEE 802.3 provided 10 Mbps (1,000,000 characters per second). Later versions dropped the CSMA/CD mechanism in favour of full-duplex "Switched Ethernet", as well as introducing 100 Mbps "Fast Ethernet" (now the most commonly-used standard) and 1 Gbps+ versions for both copper and fibre-optic transmission.

Figure A3.2 shows the Ethernet protocols together with the upper layers of the TCP/IP stack. While 802.3 specifies the media access protocols, the linkage between these and the IP network layer prompted the inclusion of a "Logical Link Control" (LLC) layer. The LLC protocols have not received much attention over the years, but still allow a frame to specify what type of packet payload it carries (IP, IPX, etc.). This thesis mostly concerns the lower MAC layer.

IEEE 802.1	Bridging (networking) and Network Management
IEEE 802.2	Logical link control (upper part of data link layer)
IEEE 802.3	Ethernet (CSMA/CD)
IEEE 802.4	Token bus (disbanded)
IEEE 802.5	Defines the MAC layer for a Token Ring (inactive)
IEEE 802.6	Metropolitan Area Networks (disbanded)
IEEE 802.7	Broadband LAN using Coaxial Cable (disbanded)
IEEE 802.8	Fiber Optic TAG (disbanded)
IEEE 802.9	Integrated Services LAN (disbanded)
IEEE 802.10	Interoperable LAN Security (disbanded)
IEEE 802.11	Wireless LAN & Mesh (Wi-Fi certification)
IEEE 802.12	demand priority (disbanded)
IEEE 802.13	Not Used
IEEE 802.14	Cable modems (disbanded)
IEEE 802.15	Wireless PAN
IEEE 802.15.1	(Bluetooth certification)
IEEE 802.15.4	(ZigBee certification)
IEEE 802.16	Broadband Wireless Access (WiMAX certification)
IEEE 802.16e	(Mobile) Broadband Wireless Access
IEEE 802.17	Resilient packet ring
IEEE 802.18	Radio Regulatory TAG
IEEE 802.19	Coexistence TAG
IEEE 802.20	Mobile Broadband Wireless Access
IEEE 802.21	Media Independent Handoff
IEEE 802.22	Wireless Regional Area Network

Table A3.1 List of IEEE 802 working groups

Table A3.1 shows a breakdown of the IEEE 802 working group into the different sub-groups, each of which is responsible for a different Ethernet standard. These sub-groups cover WiFi, Personal Area Networks (Zigbee and Bluetooth), WiMAX and many other protocols. Technically the IEEE is only responsible for specifying protocols, electronics and signalling standards. The "raw media" standards are maintained by the Telecoms

Industry Association and Electronics Industry Alliance TIA/EIA-568, which specifies fibre cabling quality.

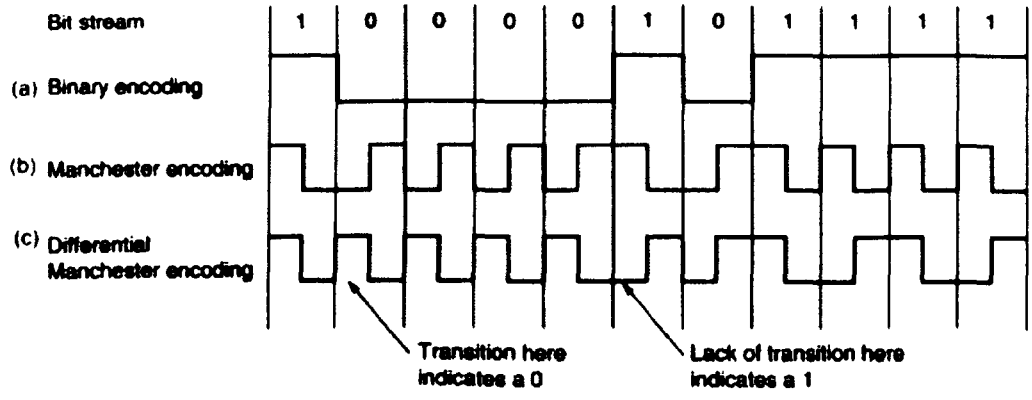


Figure A3.2 Illustration of Manchester Coding (Herald 2003).

3.3.1. Manchester Encoding

In the early days, binary one's and zero's were represented directly by voltage levels on the line, but these suffered from poor noise immunity. For this reason Manchester Encoding is now used. Here a transition from low-to-high voltage level represents a logical 1 and a high-to-low transition represents a logical 0 (see Figure A3.3).

3.3.2. Ethernet Frame Structure

The IEEE802.3 based Ethernet frame consists of preamble of 56 bits, start of the frame delimiter (8 bits), destination MAC address (48 bits), source MAC address (48 bits), the frame size (16 bits) and frame check sequence field (32 bits). The payload data field (which also contains the LLC header) has a maximum size of 1500 bytes, and a minimum size of 46 bytes. If the frame contents are below 46 bytes, padding bytes are added to bring it up to the required minimum.

Figure A3.0.2 Ethernet Frame Structure (Panko 2005)

3.4. Legacy Ethernet

Carrier Sense Multiple Access with Collision Detection (CSMA/CD) is a method for transmitting data half-duplex along a single co-axial bus, connected to all the computers on a network. Frames from all these computers are able to collide with each other on this bus, and thus form a common "collision domain". The data is transmitted by a computer when the bus is sensed to be idle. The transfer stops in the event of two stations attempting to send data at the same time. In this event a collision, the station transmits a jamming signal causing all stations to wait a random length of time before resuming transmission. Table 2.2 shows details of the two main standards for Legacy Ethernet, namely Thick Ethernet and Thin Ethernet.

Thick Ethernet (10 Base 5 or ThickNet) uses an RG-8 thick coaxial trunk cable which resembles orange garden hose. "Vampire" taps are used to penetrate the cable, with a pin inserted halfway into the core of the cable. These taps prove connect to each computer's Network Interface Card (NIC). (see Figure A3.4)

Thin Ethernet (10Base2) is similar to 10Base5, except for the co-axial cable itself, which is thinner.

10Base5 is the original Ethernet system that supports a 10Mb/s transmission rate over "thick" (10mm) coaxial cable. The "10Base5" identifier is shorthand for 10Mb/s transmission rate, the baseband form of transmission, and the 500 meter maximum supported segment length. Although when this cable is properly installed it is very reliable and easy to add new stations to (just by tapping into existing cable segments), the coaxial medium does not support higher speed Ethernet standards and (because of its

thickness and heavy weight) it is inflexible and a challenge both to installation and fault insolation.

10Base2 supports a 10Mb/s transmission rate over "thin" (5mm) coaxial cable. It is also known as "thin Ethernet", or "cheapernet". It was the first new variety of physical medium to be adopted after the original thick Ethernet standard. It is cheaper and easier to install than 10Bbase5, but because of its "daisy chain" segments, it is difficult to rout in an office environment and is not as reliable as 10Base5. It also does not support the higher speed Ethernet standards(TechFest 1999).

Technical Name	**Cable/Wire type**	**Max. Segment/wire Length**	**Maximum number of Nodes/Segment**	**Advantages**	**Minimum Spacing Between Stations**	**Ohms impedance**
10Base5	Thick (10mm) coaxial	500 meters	100	Long cable length	50 meters	50 ± 2
10Base2	RG58 (Thin5mm) coaxial	185 meters	30	Low cost	0.5 meters	50 ± 2

Table 0.1 Wiring and cabling standards of 802.3 (TechFest 1999)

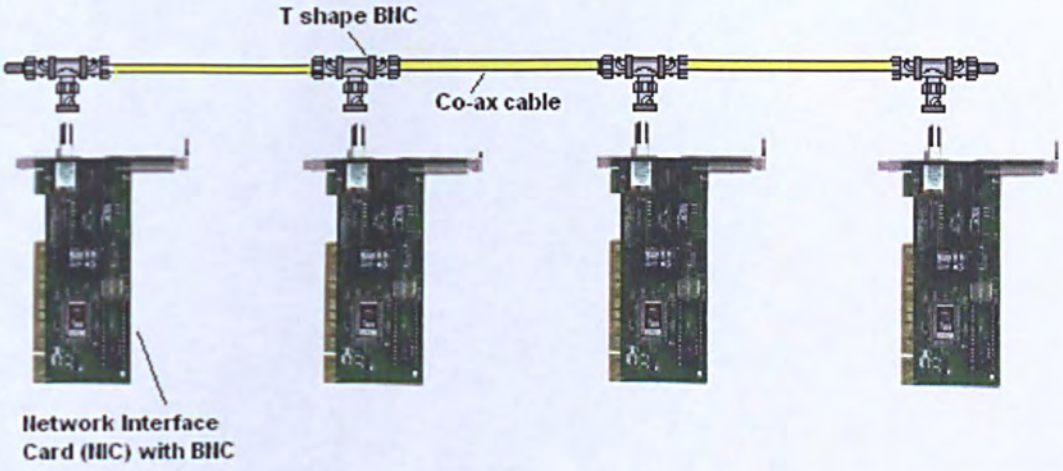


Figure A3.0.3 Interconnection of NIC cards to a common bus from (Herald 2003).

The ends of the bus need to have matched terminations to prevent frame duplication by reflection. Also as electrical impulses travel, they lose energy and the signal becomes weaker. For this reason the longest cable (segment) length is 500m, although multiple segments can be connected by repeaters (devices which restore the signals to their original strength). However, no more than four repeaters can be employed, since otherwise the round-trip time may exceed the time duration of a frame (Herald 2003).