Data Cycling in Networks: Thoughts and Experiments

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Vaibhav Desai

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

Prof. Marc Riedel and Prof. Zhi-Li Zhang

September, 2016

Acknowledgements

Many thanks are due to Prof. Marc Riedel, who constantly encouraged and advised me to work on and write this thesis since we first discussed the idea. I'm grateful to him for all the interesting discussions and pointers during the period of this work.

Many thanks are also due to Prof. Zhi-Li Zhang for agreeing to co-advise me and for all the meaningful and productive discussions. I would also like to thank him for providing me an opportunity to work on other projects.

I express my gratitude to my academic advisor Prof. John Sartori for agreeing to serve on my review committee and for supporting me through my graduate school.

I would like to thank Carlos, Chimai and staff at ECE who have enthusiastically coordinated and helped me during the course of my graduate school and all my projects.

Last but not the least, I am ever grateful to my loving parents who have encouraged, inspired, taught and pushed me to be the best in all my endeavours. Their work ethic and open-mindedness has always been a constant source of inspiration for me. I would also like to thank them for motivating and supporting me to attend graduate school, which would have otherwise not happened.

Dedication

To Margie. *"Action* $\rightarrow$ *Reaction"*.

Abstract

Communication systems rely on underlying networks and networking infrastructure to send information from one participating node to another. In this process of data transmission, the sent information experiences delays in the network. The amount of delay in network depends on the underlying network , the processing elements (such as routers, switches etc.) and time taken for physical transmission . These delays can be thought of as constituted by various paths that transmitted data takes before it arrives at its destination. The delays "emulate" temporary data storage. This study explores and analyzes this *transient storage* in different types of networks. The study introduces transient storage and data cycling from the perspective of physics and electro-optical computer networks.

The idea of transient storage caused by network delays and temporary network buffers is used to build a transient storage system by creating loops of networked nodes. Data can be cycled on loops for temporary storage before being used. A loop of nodes is used to demonstrate the idea of *data cycling* and transient storage. Data cycling experiments are setup to characterize it and to study its effects on network traffic. Further, applications of transient storage are discussed. An application of transient data storage for accumulative counting in vehicular networks is developed and analyzed in detail. This application is implemented and simulated in a network simulator to study its performance and sensitivity to variation of tunable parameters.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Evolution of Packet Switching

Back in the 1960s, the concepts which todays networks use were being tried and tested in small settings. Circuit switched networks were the standard for all telecommunications and data, which preallocates fixed communication resources. Dynamic allocation of resources was its counterpart. In 1967, Donald Davies of UK's National Physical Laboratories, suggested the use of his version of "Packet Switching" for ARPANET (DoD's fancy network)[10]. The idea was great (but not new!). Instead of having a dedicated connection (circuit) for routing data to its destination, each packet would find its own way independent of others in packet switching. Communication resources are allocated (or shared) on-demand, dynamically, for only as long as it required. This is the basis of packet switching.

Circuit switched networks were tested and well established for all telecommunication purposes. The need for dynamic allocation and ideas of better resource usage came from outside telephony, in particular from a new requirement: to connect computers for data exchange. To do this communication networks required two key components. One, store and forward capability (buffers) at every switch. Second, every switch needed processing power to perform computations for routing [30]. It was a tough sell and most public network companies, including AT&T, did not want to introduce new packet switching capabilities.

However, ARPANET was completed to demonstate packet switching networks, first with 4 nodes and 50 kbits/sec leased lines. The network was quickly expanded to 60+ nodes by 1974. Every node on ARPANET was a microcomputer connected by the same 50 kbits/sec leased lines. The system was in essence fully distributed. Each of these nodes exchanged data as 128 byte packets. When a node received a packet, it would strip the existing header and add a new header. According to a locally maintained, dynamically changing routing table, the next node enroute was added as the destination and sent out on the fastest connection to the final destination. The receiving node would acknowledge this packet and would repeat this process [30]. The success of ARPANET alleviated the massive skepticism towards packet switching.

Most early packet switched networks were private and were available only to closed groups of users. Public packet switching started in 1976 with the standard X.25. X.25 was a packet switched network standard which allowed for establishing upto 4095 virtual circuits (VC) over a duplex wire. Each virtual circuit was an end-to-end connection that was setup on demand by a setup procedure. It was torn down when the connection was no longer required. X.25 had some advantages with regard to flow control of individual VCs independent of others. This prevented any one rogue computer from sending more than what the network could handle.

Next, in 1974 Bob Kahn while working on a Satellite packet network project, came up with initial ideas for Transmission Control Protocol (TCP). Joined later by Vint Cerf, they invented a protocol suite that we know today as TCP/IP. Some functionalities were moved to a lower layer in order to distinguish functionality and separate them in different “layers”. The new layer was named the Internet Protocol (IP). Together TCP and IP formed a “protocol suite”.

This progress single handedly changed the scenario in favor of packet switched networks. “Reliable” connection and assured packet delivery was now possible on unreliable networks. The early computer industry also influenced further development of packet switched core networks. With a based dial-up connection based last leg link (home to nearest exchange) individual users could access packet switched ”Internet”. The evolution and density of networking hasn’t slowed down since. Inter-networking, be it the core network, exchange to local wide area network (WAN) or the last leg local area network (LAN), have rapidly evolved and become denser and faster.

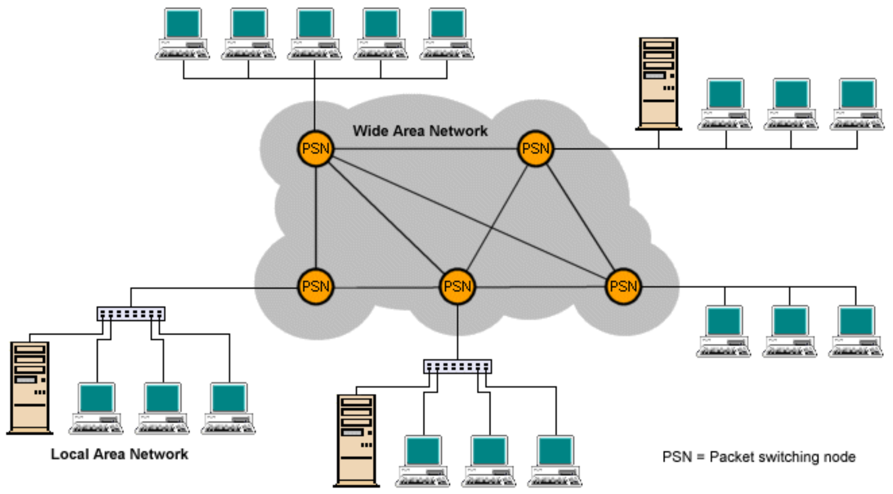


Figure 1.1: A typical packet switched internet segment [33]

1.2 Today's Internet and Observations

Figure 1.1 shows a typical organization in todays internet. Most endpoint hosts (like laptops or phones connected to WiFi) rely on some kind of local network. Ethernet LANs are pretty common. More common now is wireless LAN (WLAN) which enable millions of "smart" devices to connect to internet. With ever multiplying number of users/hosts, the packet core network has evolved in density and speed. Below we quickly analyze the physical lines that enable worldwide high speed internet.

The figures 1.2 and 1.3 show some fiber optic internet backbones. These are optical cables employing some form of dense wavelength division multiplexing (DWDM) with speeds up to 40 Gbits/sec (about 38.5 Gbits/sec of payload - OC-768) per fiber optic pair. Typically only one fiber of the fiber pair is used. The other provides redundancy in case of failures. Over this ISP (Internet Service Provider) backbone, there is a distribution network connecting ISP backbones to local point-of-presence (PoP - telecom exchange or a switching node). This is in general fiber optic too. Last mile connectivity is either one of: coaxial cable, twisted pair, ethernet or an optical fiber. Given the

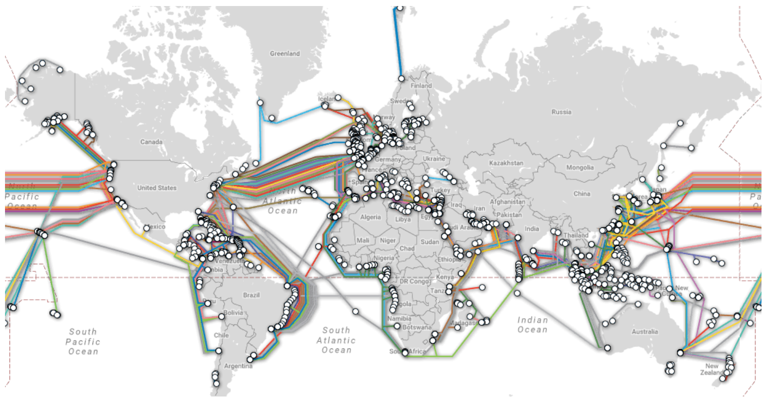


Figure 1.2: Submarine fiber optic cables for intercontinental communication [32]

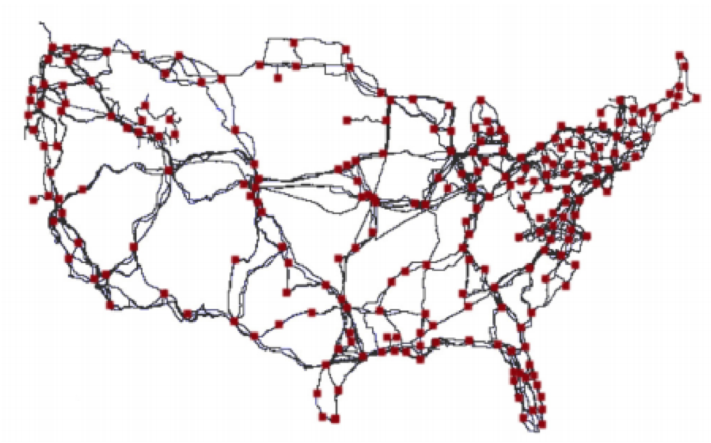


Figure 1.3: Fiber optic long-haul network in the US [11]

number of devices connected to internet today, the density of internet is quite enormous.

In the previous section the requirements for packet switching were discussed as: store and forward capability and processing power at each node. This is true for all nodes on internet. Furthermore, every communication line has an inherent "delay" associated with it. Imagine the internet transporting over 100 Petabytes of data per hour and for an instant let's "pause" time to analyze. *Where is all that data physically?* The answer is: all that data is being resides on transmission lines, or is being processed by network nodes or is in the data buffers inside switches (or other network elements)! This data, which isn't hasn't reached its destination yet, will be termed *transient data* in the rest of this thesis. In this study, the nature transient data is analyzed from the perspective of storage, quantity, retrieval speed, and some possible applications of using transient data are explored. Further, we also explore forming loops of network nodes to constantly forward data from one node to the next, essentially maintaining transient data in a *data cycle*. The work provides an introductory analyses of such loops and temporally contextual nature of transient data.

Next section provides an overview of this work.

1.3 Scope of study

This work is organized as below.

- **Chapter 2** elaborates the idea of *transient data* and discusses details from the perspective of physics of communication.
- **Chapter 3** discusses details of the idea from a computer networks stand point and explores the nature of transient data in packet switched networks.
- **Chapter 4** presents a basic implementation of a *data cycle* using a loop of network nodes and some experiments on it. Uses and limitations are discussed.
- **Chapter 5** discusses the intricacies and challenges of transient data cycles and loops on wireless networks. Further the idea of using transient data is explored on adhoc networks (vehicular networks). A counting application is presented which exploits the temporal properties of transient data. The application is simulated on NS-3 and parameter variation experiments are presented.
- **Chapter 6** summarized and concludes the presented ideas and provides possibilities for future work.

Chapter 2

Physics Perspective

In this chapter the idea of transient data and creating cycles in communication networks are from discussed from a very basic physical perspective. The below experiment presents these ideas.

2.1 The Garden Hose Experiment

Alice and Bob have their private ingenious communication system which they've spent time building and it is their only mode of communication. They now use it for all their communication when they are both home (as shown in Fig. 2.1). They have a pair tubes bundled together that run between the two houses. The pair means that there are two independent tubes inside the hose. Each tube is used in one unique direction for pushing messages written on paper. The piece of paper is rolled and pushed into the tube. They've also build the end receive/send boxes. These boxes hold the messages shortly before they are injected into the tubes. The boxes are equipped with a vacuum push-pull mechanism which is used to push out a pending message into the outgoing tube and pull messages from the incoming tube (if there is any sent from the other side).

Figure 2.2 shows how the box-tube system works. The idea here is to make sure that a path exists for full duplex communication (simultaneous bidirectional communication).

If Alice wants to send out a message to Bob, she writes the message on a piece of paper, rolls it and places it in her outbox. This message is pulled into the tube and after a while appears in Bob's inbox. Bob later picks it up from the inbox. It works similarly

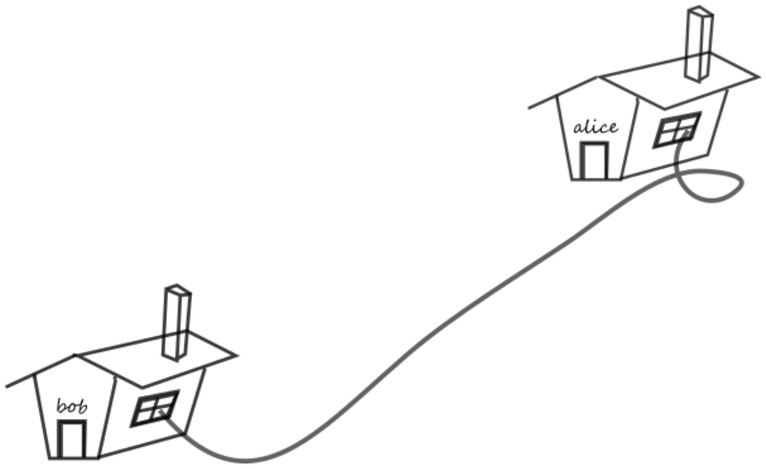


Figure 2.1: Alice and Bob's garden hose communication system

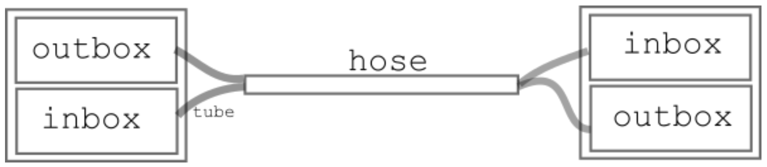


Figure 2.2: Garden hose system

in the other direction. It takes t_{ab} time for the message sent by Alice to get to Bob's inbox. For completeness, let's say Alice and Bob leave the messages in their inbox and outbox only as long as they need to be there, i.e, the minimum time. They are removed as soon as reception is complete. For simplicity we assume both times (time for which messages stay in inbox and outbox) are the same, and call this time t_p, the processing time. We can also assume that delays in both directions are equal. In other words, t_{ab} $= t_{ba}$. So, the "delay" associated with one simple message exchange is:

$$t_{delay} = t_{ab} + t_p \tag{2.1}$$

For this time t_{delay}, the sent message is in "transience", not available for access. The longer the message is in transience, the longer is the communication delay. For most communication systems, one of main performance metric is reducing this tranmission delay, to make communication faster. The lower this delay is, the faster sending and receiving is and messages are stuck for lesser time in the communication system.

However, notice that both times are finite and hence constitute to a non-zero t_{delay}. This delay is very interesting. Theoretically, what this means is, the longer the delay t_{delay}, the longer the message isn't available, the longer it is in transience. This also means longer the message is held in the system or temporarily stored. Although this isn't the intention of a communication system (in fact it is the opposite - minimize delay or holdup), it is a very noticeable property of communication systems. Alice and Bob notice this too. So they decided to play with and test the system. They decided to not remove messages from outbox on reception, but instead, modify their "terminal box" slightly by connecting their inbox to outbox.

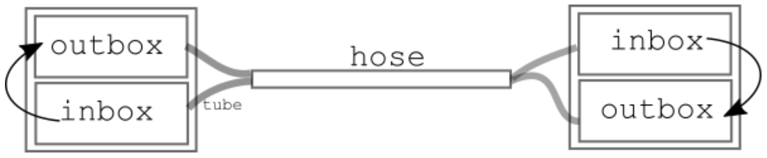


Figure 2.3: Modified garden hose system

This would allow incoming messages to be sent back to the sender, hence creating a *loop* or a *cycle*. This modified system is interesting because, apart from time t_{delay} taken for transmission, now messages start becoming available periodically to Alice and Bob (every t_{delay} time). If they want to read any message, they could simply intercept them periodically. This also means, they could just leave their messages to continuously cycle on the hoses instead of storing them separately upon reception.

The delay t_p now includes the time taken for intermediate processing when moving messages from inbox to outbox. The timing diagram shows the modified communication cycle. t_p has been assumed to be the same at all exchanges.

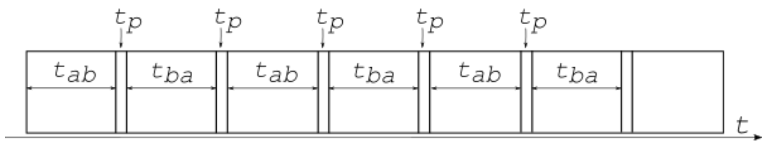


Figure 2.4: Alice and Bob's communication cycle

Now to extend their communications, Alice and Bob invite Carol to join them. Carol installs a similar modified box along with hoses going to both Alice and Bob. Now they have a three way communication system. What is also possible is that each one of them can be reached by any other via a third person. For example Carol can receive her messages from Alice through Bob, who forwards them to her (assuming this is possible just by looking at the addressee). To allow this, they all now have a new store and forward mechanism in their boxes.

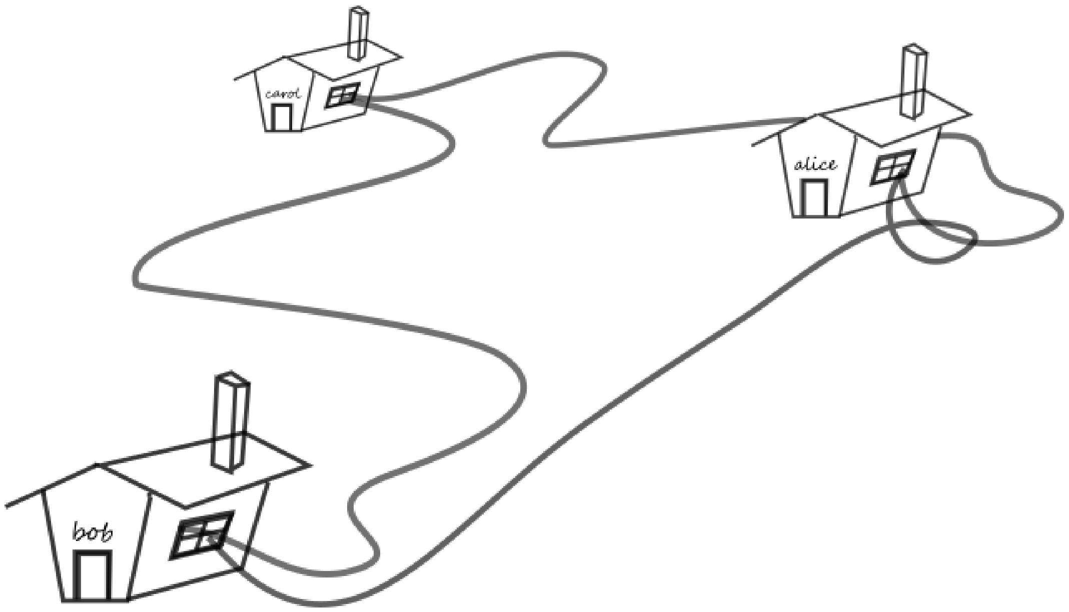


Figure 2.5: Alice, Bob and Carol communicate using garden hoses

The time taken for messages to reach each person changes as consider three different quantities need to be considered. Alice to Bob takes t_{ab} time, Bob to Carol takes t_{bc} and between Carol and Alice it takes t_{ca} time. We still assume the same to and fro symmetrical delays. That is, $t_{ab} = t_{ba}$ and so on. Figure 2.6 shows the new timing on this communication system. The periodicity changes to:

$$Period = P = t_{ab} + t_{bc} + t_{ca} + 3t_p \tag{2.2}$$

Note that processing time t_p is inclusive of push-pull and inbox-to-outbox handling delays. We formally define *wait-time* as the time for which each person has to wait to access or read a message once it has been forwarded on the loop (by the same person).

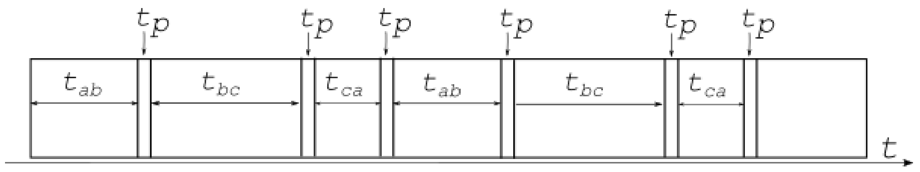


Figure 2.6: Alice, Bob and Carol's communication cycle

Since we know the *period*, wait-time is just one t_p shorter (since no wait is necessary for processing time).

$$waittime = W_t = t_{ab} + t_{bc} + t_{ca} + 2t_p = P - t_p \tag{2.3}$$

Now the next obvious (or logical) question is: *How many such messages can we keep cycling?* Since we are using garden hoses as the medium for sending messages, these can be packed to their volume limits. More concretely, if the length of each rolled message is L_m and the length of the tube is L_t then simply dividing L_t by L_m would give us the limit for a tube (assuming only one roll can be inserted at a time). Since we have two tubes in every hose, the limit will be two times as much. If all the hose segments are full such that no new message can be pushed in, then the system reaches its *cycling limit*. Note here that we have a system which does not slow down due to higher occupancy. Beyond this point, the communication system breaks down and is rendered inoperable.

However, if operated under limits and by continuous removal of *"old"* messages which are of less importance (when and if new messages are introduced), the system can be sustained well. Clearly, the limits and sustainance method of different communication systems can be very different from each other. But, as long as the store-and-forward and delay properties hold, the possibility does exist. To summarize the main points of this section:

1. By creating *loops* or *cycles*, data can be maintained on a communication network by using its physical delay and store-and-forward buffers (transient data).

2. Such a *cycle* can have periodicity associated with accessibility of required data.

3. Such a *cycle* will also have an upper limit for the amount of data that can be cycled.

In the next section, we will consider more realistic data exchange and communication networks and compute their delays, availability and cycling limits.

2.2 Parallel Mirrors

Light, in one form or another, is probably the most studied electromagnetic radiation. Light also places physical universal upper bound on speed of communication (speed of light). More importantly, light is a very widely preferred choice for data communication around the world. Optical fiber communication has revolutionized high speed backbone links and broadband like no other technology. Light, like other electromagnetic radiation (Eg. Microwaves) can be used as a high frequency carrier for data communication. The fundamental concept of modulating light with required data for transmission in optical fibers has remained the same over years[18] although there has been a significant advancement in fiber optic technology over the past decade.

In this section we try to get a sense of transient data quantitatively. Although, the quantities calculated and presented here are not indicative of the exact amount of storable transient data (this varies with network types as is discussed in the next chapter), it gives us an intuition of how transient data might be trapped and stored on networks. A small data looping experiment is presented using modulated light carrying data. The quantity of trapped data is computed and analyzed. Before jumping into the experiment, a small background on modulation techniques used in optical communication is presented.

Many modulation schemes exist to modulate carrier light from sources such as Lasers or LEDs to carry data. Modulation schemes have evolved from basic binary on-off keying (OOK) to differential phase-shift keying (DPSK) and now most systems have migrated to wavelength division multiplexing (WDM) and dense wavelength division multiplexing (DWDM) for high spectral efficiency [20]. What this means is, more data can be effectively packed per wavelength/carrier frequency than ever before. Detection devices have also been enhanced to detect these modulated carriers with very low bit error rates (BER). A comprehensive coverage of DWDM, optical scattering effects impeding it [6] is out of the scope of this work, but some details of DWDM are used in the next section

. Of a greater interest is the question: *How much data can light be modulated to carry?* Researchers have shown that optical modulators can be built to support high speed applications and can easily modulate light to carry 40 Gbits/sec of data [24].

Over the last decade or so, visible light communication (VLC) has developed as a viable method of communication too. VLC has been studied quite in detail and many theoretical analyses using LEDs have shown possible communication speeds upto 10 Gbits/sec [22]. Although still in its infancy, some prototypes have already demonstrated VLC exceeding 500 Mbits/sec [35].

To set up the apparatus for the experiment, let us consider two *perfect mirrors* separated by one meter such that the reflective surfaces are facing each other. The mirrors are square with a side *1 meter*. These are placed in a completely evacuated chamber (vacuum between mirrors). Vacuum renders the space between mirrors (medium) non-dispersive and non-scattering allowing for lossless propagation. Figure 2.7 shows the arrangement of mirrors as seen from top. The arrangement has a striking similarity to Fabry-Pérot interferometers [17], but is on a much larger scale and is being used for a different purpose.

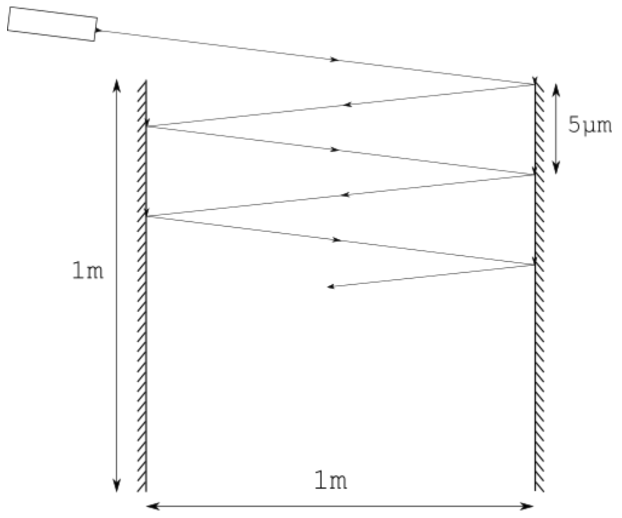


Figure 2.7: Top view: Parallel mirror apparatus

The edges of mirrors are special. They have special micromirrors which can reflect light vertically to move reflections to a next reflection plane. The arrangement is such that when light has been reflected back and forth through the full length (say along the width) of the mirrors, it is moved (by reflection) to the next plane by the micromirrors and the plane part of mirror then reflects light back and forth again, but the beam travels in the opposite direction along the width. Once it reaches the other end, it is moved to the plane below it. This process continues until the full height of the mirror is used. In summary, we are attempting to "fold" a beam of light as much as possible in a cube of size *1m*×*1m*×*1m*. Figure 2.8 shows the placement of micromirrors and reflection planes.

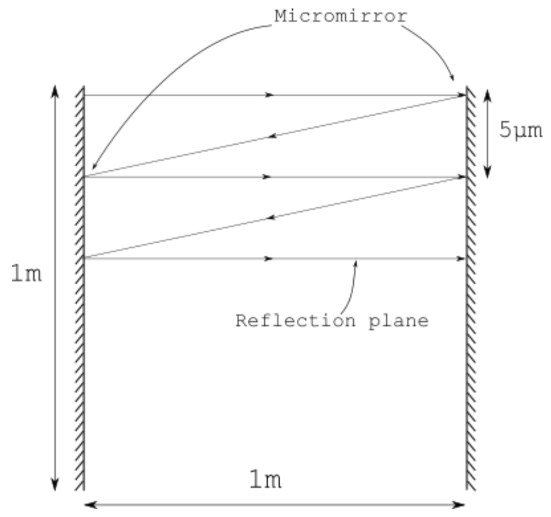


Figure 2.8: Side view showing reflection planes and micromirror positions

The below assumptions are made to simplify the optical details in the experiment.

1. *Perfect mirrors*: In the physical world that we live in, there are no perfect mirrors. The best of them have reflectivity ranging between 97% and 99.99%, the latter being called supermirrors. However, since this is more an analytical experiment and not a prototype apparatus, we can make the assumption. Another option for 100% reflectivity is to use a phenomenon called *total internal reflection* which is the concept used to build optical fibers. But to simplify our experiment, we consider perfect mirrors.

2. *Micromirrors*: Microelectromechanical systems (MEMS) micromirrors are widely

available and are used in light processing devices ranging from projectors to optical switches. It is very well possible to construct a large array of micromirrors. Micromirrors of the order of 3μm arranged as hinged arrays have been produced commercially [39]. The assumption of 5μm spacing is a valid one.

3. *Diffraction and divergence*: Beam divergence is a big problem when dealing with collimated laser beams or other narrow light beams. Longer wavelengths and narrower beams tend to diverge quickly than their counterparts. One way to alleviate this problem is to re-focus each beam using concave mirrors. Concretely, every reflecting point on the mirror should be a MEMS concave mirror instead of a planar non-MEMS flat mirror. Each concave mirror would try to slightly converge the diverging light beam on to a concave mirror on the opposite side. The other option to avoid divergence is to use waveguides, optical fibers being the best of the options (instead of free propagation).

With this experimental setup, we can calculate what length of light beam can be *folded or trapped* in this cube of size *1m×1m×1m.* A straightforward approach can be used to calculate it. First, we compute the combined length of all folded beam segments for one reflection plane. Next, we can simply multiply this number with the number of reflection planes and get the total length.

From Figure 2.9 we can easily compute the side c for the right triangle with sides *a, b and c.* We already know $a = 2.5\mu$m and $b = 1m$, hence,

$$c = \sqrt{a^2 + b^2} = \sqrt{6.25 \times 10^{-12} + 1} \approx 1$$

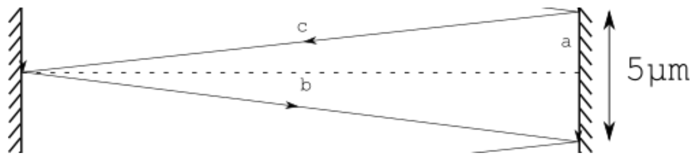


Figure 2.9: Length of reflected beam is the length of hypotenuse

For every 5μm along the edge of mirrors, there are two reflections. So the approximate distance traversed for every 5μm is $2c = 2$ meters. For the full mirror length, this is:

$$L_p = \frac{1}{5 \times 10^{-6}} \times 2 = 4 \times 10^5 \ meters \tag{2.4}$$

The number of reflection planes in the setup is simply the height of mirrors divided by inter-plane spacing plus one.

$$P = \frac{1}{5 \times 10^{-6}} + 1 \approx 2 \times 10^5 \tag{2.5}$$

The total folding length (L_t)is the product of L_p and P.

$$L_t = L_p \times P = 8 \times 10^{10} \ meters \tag{2.6}$$

With the total folding length L_t known, the amount of time light takes to traverse that distance T_f is given by Eq. 2.8 where C is the speed of light in vacuum.

$$T_f = \frac{L_t}{C} = \frac{8 \times 10^{10}}{2.997 \times 10^8} \approx 266.933 \ seconds \tag{2.7}$$

If the light beam were modulated to carry B bits/sec (bitrate), then the apparatus traps or folds light carrying data D_t, which is given by Eq. **??** at any instant of time. Note: this is true only after all the possible paths are traversed by the incoming light beam.

$$D_t = T_f \times B = \frac{266.933 \times 40 \times 10^9}{8} = 1.334 \times 10^{12} = 1.334 \ terabytes \tag{2.8}$$

If this beam was one that was modulated with DWDM to carry 40 Gbits/sec (OC-768 standard used in longhaul optical fiber communication), then, the amount of trapped data is approximately *1.33 terabytes*! We could easily create a loop here by connecting the outgoing beam back to the input again. This can be done easily by placing two plane mirrors to reflect the outgoing beam by two right angle (90°) reflections back to the top to start again. Once done, this will be a loop holding T_f seconds of light. If full and if all the components and connectors inside the apparatus were lossless, 1.33 terabytes of data could be trapped inside our apparatus!

Obviously, this would be a very expensive apparatus and would consume energy for beam maintanence if the assumptions were removed. But, the possibility of loops (or cycles) being capable of holding large amounts of data can be clearly seen. In the next section, we go study optical buffers for completeness before moving on to such loops in computer networks.

2.3 Optical Buffers

A good part of previous section's topic forms the basis for implementation of *optical buffers*. Formally, optical buffers are devices used to store *light* for a period of time before being transmitted or used in optical networks. There are many reasons which necessitate the use of optical buffer. The biggest by far is the evolution of fiber optical communication networks. The reason is subtly the speed at which modern optical networks operate. With such high speeds (over 1 Tbits/sec), conversion to electrical signals and electrical buffering becomes a challenge. Hence, a mechanism is needed to handle optical signals in their native form without any conversions, that is, buffer or store the optical signals directly. Handling optical traffic for switching right data to their correct destinations is also a challenge due to such high data rates. Imagine, at a backbone longhaul switch, two incoming high capacity fiber optic lines contending for the same outgoing line. The switch has to resolve this contention and deal with one line at a time. In such a case, data on the second line is *buffered or delayed* until, it can be successfully switched. If no buffering is possible, then the optical signals on one of those incoming lines is lost and the switch becomes a bandwidth bottleneck. To resolve this problem there has been a lot of advancement in optical switching (especially using MEMS) and more importantly optical buffering [42, 36, 25].

Optical buffering has been tried and implemented. There are currently at least 4 ways of implementing optical buffering without performance degradation at varying costs [13]. Broadly the methods used are: wavelength conversion, deflection routing and delay line buffering. The first two can have performance issues when used by themselves and hence always require some kind of buffering [13]. Optical buffering (using no tricks, but real delay lines), can be done in multiple ways too. Delays in optical fibers are complicated and depend on wavelength (chromatic dispersion), temperature and the length of the fiber [41]. Although we will not study these variations in detail, we shall glance through optical cross connect (OXC) switches and delay lines. Next, we will take a look at some optical buffers and draw analogies to the discussion in the previous section.

Optical buffers based on optical delay lines (ODL) are the most common and widely used types of buffers. Optical delay lines are simply fixed length optical fibers which act as first-in first-out (FIFO) buffers [43]. Research has also shown that silicon based coiled optical waveguides can be produced using smaller footprints than actual optical fibers in ODLs. One of them was successful enough to coil about large lengths (order of 100s of meters) of optical waveguides using lithography to produce very small footprint wafer delay lines [23]. Optical buffers can be broadly categorized into two types. Feed forward buffers and feedback buffers. The difference between feed-forward and feedback comes in how data (or packets) are handled when they leave delay lines. In feedback delay line buffers, packets arrive back at the switch which buffered them to resolve contention again. In feed-forward buffers, delay lines are setup at the output port of the switch. So once the packets finish the delay loops, they exit the switch.

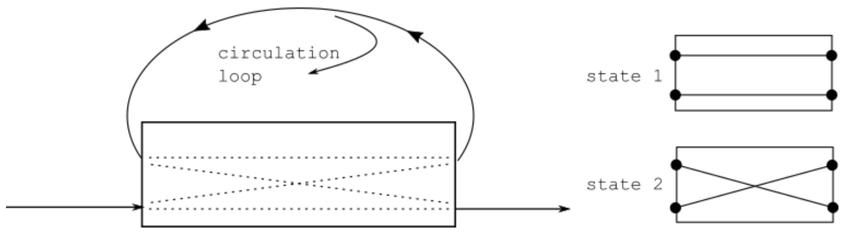


Figure 2.10: A simple 2×2 optical switch with its two states

Figure 2.10 shows the arrangement of a delay line on a simple 2×2 optical switch. The circulation (or recirculation) loop is an optical fiber which acts as the delay line. To delay packets, the switch initially in state 1 (parallel), switches to the crossed state (state 2). While crossed, the input line is connected to the circulation loop and the switch remains in this state until all the packets that need delaying are inserted into the circulation loop. Once done, it switches back to state 1 to normal switch packets. When the delayed packets on the circulation loop traverse the full delay line and are available at the switching port, the switch goes to state 2 again connecting the output of delay line buffer to the output port of the optical switch. In parallel, it would insert other required packets into delay line.

If however the switch was not ready to switch the delay line to output (possibly because it resolved a new contention in favor of the incoming line), then the packets in the delay line will be kept there and allowed to circulate again. If the length of the delay line is D, then variable delay is implemented by recirculation using the same loop in multiples of D (1×D, 2×D, 3×D etc.). In general, in high performance switching for better throughput these delays have to be kept minimal. One of the basic requirements (or goal) of most research is to shorten this delay and eliminate them where possible. But, by the definition of *switching*, when packets from different sources arrive at a switch intended towards the same destination such delays are always necessary.

By analogy to the previous discussion on parallel mirrors, an optical delay line takes the role of the parallel mirror apparatus. Although ODLs are used for delays of the order of *milliseconds*, in aggregate over large and dense networks, these delays become significant thus giving us the property (store-and-forward) that we are looking for. The data that is stuck in delay lines or in optical fibers during tranmission will contribute towards large transient data. In fact with most electrical switches being replaced by their optical counterparts, *transient data* in optical networks might some day be larger than that in electrical networks.

2.4 Summary

In this chapter, we discussed the idea of *transient data*, its availability and wait-time properties. The quantity and availability of transient storage in real-world optical networks was also discussed from a conceptual point of view. Optical buffers were introduced and shown to buffer data optically in networks. In the next chapter, we will consider transient data in computer networks and the theory associated with such data in electrical networks.

Chapter 3

Computer Networks Perspective

The main theme around which packet switching revolves is resource sharing. Specifically, *resource* here means cables, switches, hubs and all the other equipment on a packet switching network. To understand why data looping or cycling could work and what impacts it can possibly have on existing networks, concepts behind resource sharing need to be studied. Massive computer networks like *internet* work by sharing large busy switching centers, data centers and dense cabling among billions of users using the same underlying sharing concepts. The most important paradigm is called *statistical multiplexing*. Although the name is quite indicative, the next section details its basic operating principles.

3.1 Statistical Multiplexing

Imagine a network required for connecting a group of 10 users to a bigger network and providing them the best possible service. If all users need a 1 Mbits/sec connection, then this network needs to be capable of handling a peak traffic of 10 Mbits/sec. *Correct?* No, not necessarily. If we take a look at how traffic flow is, we will easily see that network usage is bursty. This means that there are times when a user has data to send and there are other times when the user's connection is idling because there is no data to send. This is a very important observation. What this implies is, in aggregate network usage, over time, the bandwidth requirement can in fact be much lesser (depending on occupancy) than the peak traffic load.

If we plot the traffic at a switch for different number of users, figure 3.1 shows load with respect to time. Note how bursty the traffic is when only a couple of users are on the network. As the number of users increase, load smoothens (higher usage). For large number of users, the load envelope settles near the *expected load* for that number of users. Since each user is using the network only for a fraction of time, the expected load is far lesser than the possible peak load. This is exactly what allows installation of a network with much lesser capacity than that corresponding to peak load. Expected load is "statistically" computed for a given number of users. The network is hence "shared" or "multiplexed" by these users in time and the network statisfies their bandwidth requirements by implementing network sharing.

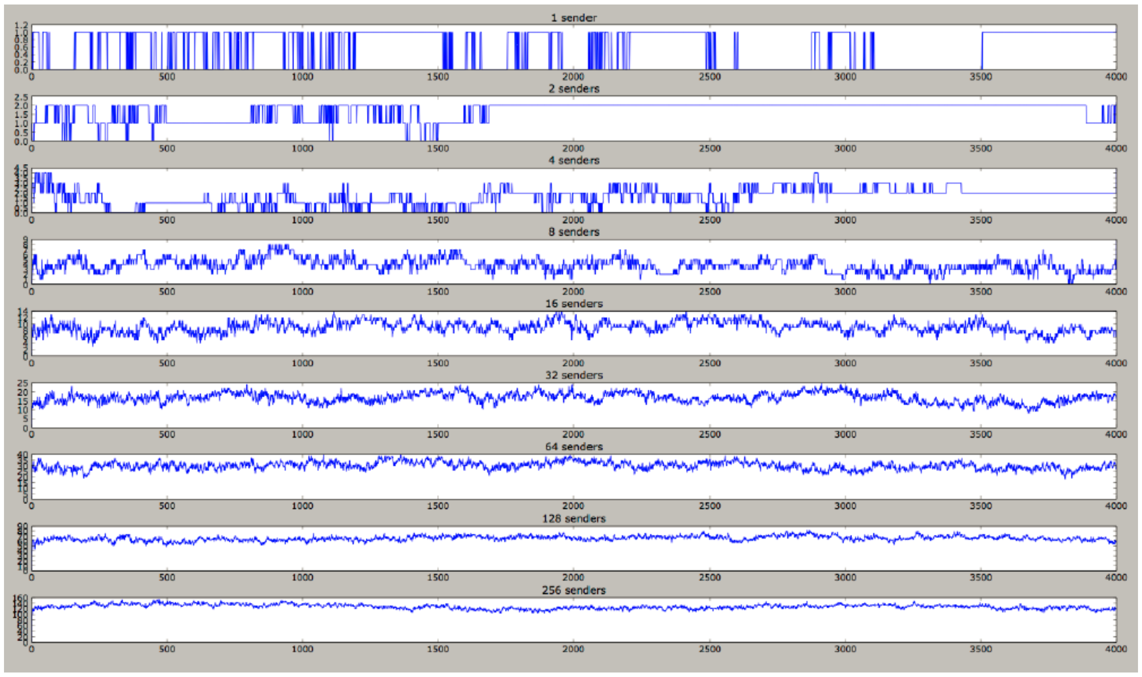


Figure 3.1: Network traffic at a switch for varying number of users vs. time [1]

An average *or statistical* undersubscription is expected when buildling a network based on statistical multiplexing. Since each user's usage time and network occupancy is random (or at least we can assume this), if we aggregate usage and occupancy over a large number of users, we can use central limit theorem (CLT) to prove that the usage will follow a Gaussian distribution. However, this calculation makes an assumption that expected values and variances are finite for each underlying sample. This assumption holds in general for network usage in packet switched networks, but there are some cases which can act as exceptions. These are called long-tail *(or heavy-tail)* distributions of traffic.

The "on-demand" channel resource allocation and sharing model used by packet switched networks also allows variable traffic rates for nodes operating at different speeds. However, this can create bursty peaks in traffic which need to handled by the switch. Switches use *queues* to absorb bursts. This is exactly where "store-and-forward" capability plays an important role in managing uneven packet arrivals and link utilization. But since networks are generally not built to handle peak load where a large number of computers are synchronized to continuously use network's link for long periods of time, such situations result in problems. However, the probability of this happening is very low unless there are malicious nodes trying to jam or flood the network. This disruption possibility opens up space for a class of network attacks called *distributed denial of service (DDoS)* where multiple malicious (or compromised) nodes flood the target node's link(s) (may be a website, file server etc.) with large number of garbage packets (also large in size) to break statistical multiplexing. It is important to note that such attacks can happen even if redundant network links are present to handle peak traffic.

As long as we maintain traffic such that statistical resource sharing of network links is working well, the networks should be able to function normally. *"Functioning normally"* can generally be attributed and credited to the queuing capability of packet switching systems.

3.2 Queuing and Network Delays

3.2.1 Queuing

Queues in a switch are necessary to handle fast arriving packets destined to go through a "not-fast-enough" outgoing link. The basic operation of a queue is simple. Packets are *buffered or queued* until they are sent out on outgoing links. If packets arrive so fast that queues fill up before they are handled and sent out, then all new packets that can't be queued are just dropped. Packet loss is an inherent property of packet switched networks and the sender has to be able to detect and handle such losses. One way this is done is by using timed acknowledgements. If the sender does not receive an acknowledgement, then the packets are re-sent. The existence of queuing in networks introduces a delay corresponding to a packet's wait time in queues. This is called *queuing delay*.

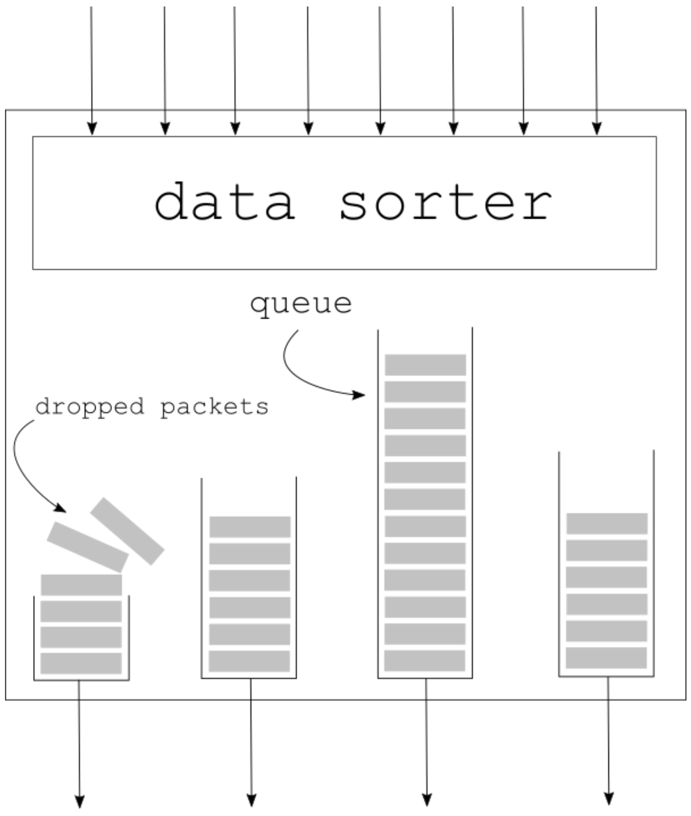


Figure 3.2: Queues in switches

The size of queues is critical to achieving good network performance. Shorter queues can end up dropping too many packets hence causing a lot of retransmission traffic. Huge queues on the other hand queue up a lot of packets and each one of them will experience longer wait time in queue before making it out. Dropping and retranmissions might win over such long queuing. A right sized queue is in fact dependent on the speed of outgoing links and will balance between queuing delays and dropped packets and hence achieve an optimal performance. Figure 3.2 gives an idea of how different queue sizes may affect outgoing packets. Data on all incoming lines are switched to four possible outgoing links. Links with large capacities can afford to have larger queues since they can bring wait times down. In general, faster the lines, bigger the buffers needed. The general rule of thumb used to size buffers (B) is the by setting them to the product of average delay (D) and bandwidth (BW) [26, 38].

$$B = BW \times D \tag{3.1}$$

So, for a link with a capacity of 1 Gbits/sec with an average round trip delay of 100ms, 100 Mbits of buffer space is needed. However, this has since been revisited and revised by many studies [2] as being an overestimate. Further, [37] provides an interesting compilation of estimated commercial routers/switches' buffer sizes. To understand the impact of cycling data in loops on networks, all possible delays in networks have to be first understood.

3.2.2 Network Delays

Packet switched networks have various *delays or latencies* associated with data transmissions. In general, reducing latencies to the maximum possible extent is a primary goal in communications research. This makes sense because, the lower the latencies, the faster we can get data to its destination and the happier we all will be.

However, network delays can be used to temporarily *store* data using cycles or loops by allowing these delays emulate temporary storage. *Storage* can be thought of as a large delay between two accesses of given data. Delays are also interesting because they attach temporal properties to given data. Here we examine the possible delays in packet networks and later discuss how these could be used to create transient storage.

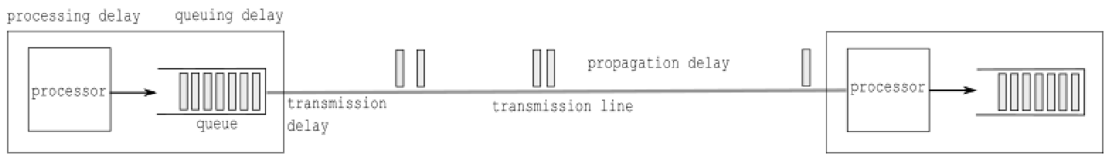


Figure 3.3: Delays in networks between two switching nodes

Figure 3.3 shows the possible delays in packet switched networks. *Processing delay (t_s)* is introduced by network processors which handle packets as they arrive at a router or a switch. This delay involves sniffing packet header to place them in the right queue and for checking bit-errors in packets and correcting them if possible using error correcting codes. Sometimes modification of headers is also necessary to redirect the packet to its destination.

Once processed, packets are placed into queues for transmission on their respective destination links. *Queuing delay (t_q)* is experienced when packets wait in queues pending transmission. Depending on the speed of links and switch configurations, queuing delay can be a significant component of the overall delay.

Next when packets are ready for transmission, they are sent on network links at speeds corresponding to link capacity. This is time taken to put a packet on a link for transmission. Given a link capacity of R bits/sec (bit-rate), and a packet of size L bits, the transmission delay (t_{tx}) is $\frac{L}{R}$. For a given link bandwidth, t_{tx} is a constant.

The packets will now take time to travel to their destination. This is limited by the physical constraints of the medium and is also dependent on the length of links. This is the *propagation delay (t_p)*. For example, on a wireless link of length 50 km, propagation delay will be the time taken by radio waves to travel 50 km (at the speed of light, 3×10^8m/s). Propagation delay in fiber optics will depend on the speed of light in glass (medium) which is approximately 2×10^8m/s (two thirds the speed of light in vacuum). Propagation delay can also be thought as the time taken for the first transmitted bit to travel from source to its destination. This is distinct from transmission delay which is the amount of time taken to send L bits over the link.

The aggregate delay between two store-and-forward switches or routers is the sum of all delays . This is also called *per-packet latency or per-packet delay (t_L)* and is an important metric to assess the performance of a network.

$$t_L = t_s + t_q + t_{tx} + t_p \tag{3.2}$$

Of the delays in Equation 3.2, t_p and t_q can vary quite a lot depending on the type of network. Either of these will dominate t_L, while t_s and t_{tx} are generally small and predictable.

Queuing delay (t_q) can infact be computed using some basic queuing theory. Imagine a queue connecting one incoming link with one outgoing link. If the average packet arrival rate on the incoming line is λ packets/sec and the average time spent by each packet in the queue is D, then the average number of packets in queue at any given time is given by the relation

$$N = \lambda \times D \tag{3.3}$$

Equation 3.3 is also popularly called *Little's Law* in queuing theory. The idea is simple. If packets arrive at a rate λ (average), then irrespective of all bursts, the queue will have an average effective delay on outgoing packets. Note that the packet departure (outgoing) rate must be equal or higher than the arrival rate, else the queue size will increase unbounded resulting in packet loss. Given link bandwidth and measured queue size, queuing delay, t_q can be computed easily.

3.3 Long Fat Networks

Long fat networks (LFN) allow large amounts of data to be *in-flight* at any given time. For maintaining transient data on a cycle, higher in-flight data means higher amount of transient storage. To understand how in-flight data can be maximized, the working of transmission control protocol (TCP) needs introduction. TCP is the most widely used reliable service abstraction built over unreliable packet switched networks. TCP forms most of web traffic. Operation of TCP is simple and it works as described in the next paragraph.

There are two kinds of broad flow control protocols in networking. First *stop-and-wait protocol* and the second is *sliding window protocol.* Simply put, in the first kind the sender waits until the receiver acknowledges reception before sending any new packets. In sliding window protocols, the receiver sends data while awaiting acknowledgement. For reasons that are clear from parallel transmission nature of the latter, it performs better and achieves near theoretical throughput in practical networks. TCP is designed to work as a sliding window protocol. The simple operating principles of TCP are:

1. Two nodes that want to communicate synchonize using a three-way handshake to start a TCP flow.

2. Once a connection is established, data can be sent in either direction. Each node maintains a sending and a receiving window. The windows are moved (or slid) on acknowledgements from peer receivers.

3. A sender can send as many bytes as allowed by the sending window. Further transmissions are allowed only on acknowledgements. Sending window size generally indicates the maximum number of bytes that are outstanding (that is, unacknowledged) at any point in time.

4. Similarly, the receive window complements the sending window. The sizes are negotiated and maintained in a closed loop so that no overflow occurs. Larger the window sizes, more the unacknowledged data.

5. If data is lost, retransmissions occur based on timers run on each side while waiting for acknowledgements. Larger the window sizes, higher the chances of big retransmissions in case of data loss.

There have been plethora of studies over last two decades on setting the correct window sizes for TCP and there by trying to maximize TCP throughput both in academia [19] and industry [4]. All the studies gyrate around a central idea called *Bandwidth-Delay Product (BDP)*. This is a useful quantity because it lets us correctly quantify the amount of unacknowledged data on a given network link. Or at least give a rough idea about it. This is very interesting as this looks exactly like the quantity we previously computed as being the transient data. BDP in fact gives a bound on how much data can be stored transiently when data is looped on a closed cycle using a computer network.

Bandwidth delay product is the product of a link's bit rate (BW, in bits/sec) and its round trip latency or round trip time (RTT).

$$BDP = BW \times RTT \tag{3.4}$$

The product gives a quantity in bits which represents the amount of data in-flight on the link and is unacknowledged (in TCP terms). It can easily be seen that longer the links, higher is its BDP. Also, higher the link capacity (or fatter the lines are), higher is their BDP. Since, we use round trip time (RTT) instead of one way delay, BDP represents total data outstanding in both directions. This leads to the notion of long fat line (LFN), which are network lines with large bandwidth delay products. The original IETF RFC-1072 [19] calls lines LFNs if BDP is much larger than 10^5 bits. Links today easily exceed this threshold. As a concrete example, for a transatlantic fiber optic line with a bandwidth of 40 Gbits/sec and a latency of 100ms, the amount of data on the line at any time is about 4 Gbits.

For the purpose of creating loops of data, targeting long hauls with higher latencies and high bandwidth looks like a good idea. Although, a lot of transient data can be stored on these LFNs, accessibility drops due to large latencies. As discussed in Chapter 1, the availability of looped data at nodes depends on link delays. But if we were to have a very large loop with strategically located nodes around the world (for example, Minneapolis → San Francisco → Tokyo → Singapore → London → New York → Minneapolis) to loop data, then we can potentially store gigabytes of transient data on these links which have both large RTTs and bandwidths.

3.4 Satellite Links

Communication satellites are of three broad types. Those in geostationary orbits (GEO), low Earth orbit (LEO) and medium Earth orbits (MEO). Out of the three geostationary ones are the most widely used satellites for communication and broadcasting. The satellites are placed in an orbit close to 36000 km from Earth's equator.

Latencies in satellite communication has been a major hinderance in its proliferation. The major component of delay is due to propagation time associated with such large distances. To simply compute the delay for a ground station to uplink and another ground station to receive downlink, radio waves have to travel about 70000 km. At the speed of light (3×10^8), the latency can be close to 250ms. Considering queuing, processing delay will increase this number by close to two times this [15]. 500ms latency is common in satellite communication, ground station to ground station (G2G).

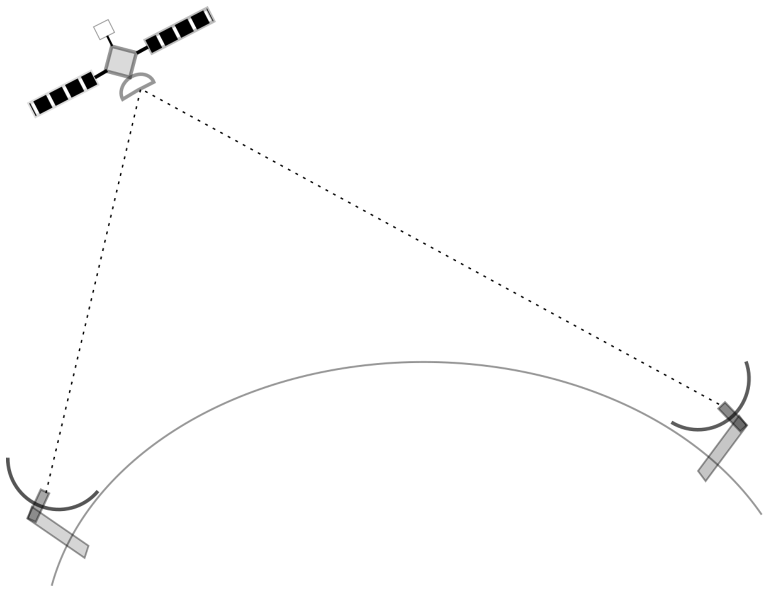


Figure 3.4: Satellite link between two distant Earth stations

A satellite link connecting two ground stations can easily be classified as an LFN. If the G2G latency is, say 500ms then round trip time of a satellite communication link is about 1 second. Even with commercial satellite internet subscriptions, 25 Mbits/sec are typical. So, multiplying these gives us a BDP of 25 Mbits. Although in comparison to fiber optics this isn't a lot, we now can see how 25 Mbits are simply in air at any given time.

To take the idea a step further, just for the purpose of thought and possibility, if we were to aim a beam of light modulated at modern capacities into outer space (say in a direction where we can't spot any stars and just see dark space), the light has a good chance of living for a very long times. But if we did want to access that information we could have some distant celestial object send it back to Earth!

3.5 Effects of looping on networks

Now that the idea of looping data on terrestial and other computer networks is discussed we can analyze the possible effects of looping data on regular network operation.

“Regular network operation” includes network accesses for web browsing, video streaming etc. The negative effects on network traffic due to looping is also considered and discussed.

Imagine four nodes on the internet form a loop and decide to store a 1 Mbit file on the network as continuous transient data instead of storing it (using conventional storage, like hard drive, flash etc.). The most obvious thing that can be noted is the increased packets exchanged on the links between these nodes. When all regular one way traffic is stopped, there will still be traffic caused by looping data, about 1 Mbits of data packets to be exact. These packets, at any given time are in queues of network elements, in propagation on network links or are under processing for transmission, influenced by various forms of latencies discussed earlier. If the links between nodes are short and have low bandwidth, this looped data will contribute to a significant performance drop for regular traffic. To explain concretely, consider 4 nodes connected via 512 Kbits/sec link where the latencies are such that 1 Mbits of looped data has to be sent every 4 seconds. Assuming other delays are insignificant, the links will be occupied 50% just by such data. However, on the contrary, as most modern high speed links are, with larger bandwidth and longer latencies (for example on a 1 Gbits/sec, 200ms link) occupancies are much lower and hence the issue is slightly alleviated.

When discussing statistical multiplexing, it was shown how shared usage of network resources is the basis for packet switching. Now, with looping usage of resources becomes higher than usual. This is again because packets are forwarded from one looping node to the next continuously hence increasing the amount of link usage by looping nodes. In effect, if the forwarding was continous, this would add an average constant value to network utilization. This can be the case on short and slow links. If forwarding is non-continuous and well spaced then looping traffic behaves as normal traffic. In fact, in the next chapter we will show through experiments that data cycling can be useful for applications using small amounts of ephemeral data and not for storage of large amounts of data.

Until now data cycling has only been discussed from the persective of looped packets. The type of protocol used and mechanism used affect how much of looped data can be reliably kept on a network. In general looping can use conventional protocols like UDP and TCP. When using UDP, losses can be significant. Recovery from such losses is not possible since large chunks of data may just disappear because of participating node outages. Redundancy can be introduced to increase reliability. However, unreliable transport protocol can be an issue even for flow control. Since UDP provides no flow control, a fast sender can easily cause buffer overflow losses in switches. UDP can still be useful when data loss is tolerable. As a concrete example, a wind speed monitor generating periodic measurement data packets can be occasionally lost without significant impact.

To summarize the discussions in this chapter, data looping is feasible when the amount of data looped is small compared to regular traffic. We have also looked at networks which have interesting present large delays and bandwidth, and are hence useful for transient storage. In the next chapters some applications of looping are discussed and looping is demonstrated using real setups and simulations.

Chapter 4

Data Cycling Experiments

Continuing on the ideas of transient data pseudo-storage in network loops introduced so far, in this chapter we implement a prototype for the purpose of qualitative and quantitative analysis. Further, we explore applications of transient data storage in networks involving small systems such as connected devices which generate new data periodically.

4.1 Experimental Setup and Objectives

4.1.1 Setup

A simple loop can be formed by connecting two nodes (like Alice and Bob) and simply resending all received data back to the sender. However, there is a good chance that both to and fro paths share the same network elements and this can result in a systematic errors on both paths if there is an issue with the path. While it can be a good setup to study, a better setup can be built with 3 nodes to form a clear clockwise or counter-clockwise cycle.

Nodes are setup as shown in Figure 4.1. The loop consists of 3 nodes, each running a cycling application. The data forwarding direction can either be clockwise or counter clockwise. For clarity we'll use node-1 $\rightarrow$ node-2 $\rightarrow$ node-3 $\rightarrow$ node-1 as the data flow path. Apart from cycling, each node also supports injecting new data into the cycle and retrieving cycling data from it.

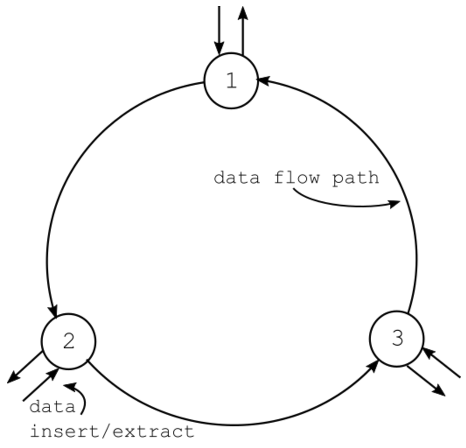


Figure 4.1: 3 Node Cycle - Experimental Setup

In effect, each node also acts as a point of access for the data on the loop. This provides a service functionality for external nodes which want to access data but are still not a part of the loop. However, the primary users of cycled data will be nodes themselves. Each node is a general purpose computer running a standard operating system and an IP stack. Here we use three inexpensive Raspberry Pi(s) as nodes. Each node is connected to a large IP network with access to internet. The path between nodes is different between each pair. The nodes are physically separated so that they are all not connected to the same network elements and to avoid very short single hop paths. Although this setup does not use LFNs, the analysis can be extended to LFNs which are better suited for cycling due to their large BDP.

Figure 4.2 shows the possible network elements and shared paths. The nodes are connected using 10/100 Mbits/sec ethernet to network. The approximate round trip times on each path is also shown relatively by the length of links in Figure 4.2 (note that this varies with external loads of network elements). Some segments of paths between node pairs go through the same routers.

Each node runs a *cycling application*, which is responsible for maintaining data on the established loop. In case the underlying transport protocol is reliable (like TCP), the application also establishes and maintains the cycling connection between the node and its cycle destination (which is the next node on the loop). The general organization of this application is as shown in 4.3.

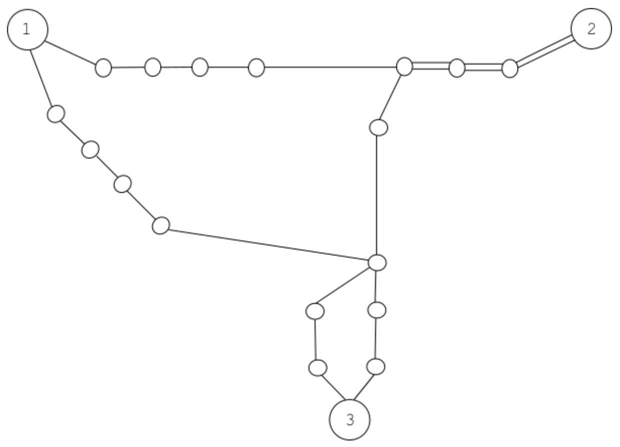


Figure 4.2: Network path between nodes

The cycling application consists of modules, each managing a particular task. The most basic is the cycle module which manages data cycling. Cycle module establishes and maintains *"next node"* connection to forward data to. Cycle module listens to incoming data from its predecessor node and forwards the same data to its successor node (primary activity). It also periodically collects data from its internal queues.

The internal data queues are managed by *insert and extract module.* These are service providers to nodes which need to put data on or retrieve data from the loop. The insert module collects external data and puts it on the loop. Complimentarily, the extract module retrieves requested data from the loop, if available. All the modules are implemented as separate threads which are managed by the controller. Insert and extract modules need not always exist. They are needed only on demand are hence created and destroyed as necessary by the controller. Apart from these functional modules, the application also has a measurement module for data collection purposes.

The cycling application is started as a daemon which runs on background. At every node when the application starts, it looks for a configuration set. The configuration set is received by each node when it joins the loop. The configuration set contains the below details:

- *Predecessor:* The address of the node from which it receives cycling data.
- *Successor:* The address of the node to which it sends data.

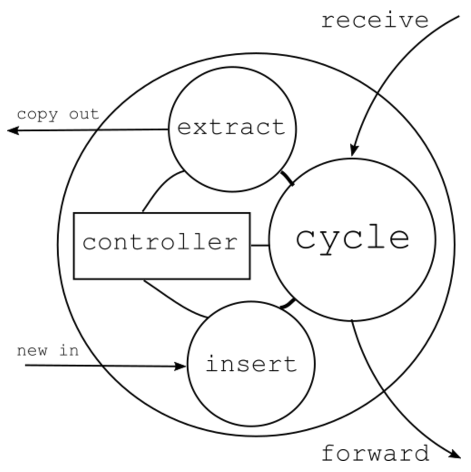


Figure 4.3: Elements of cycling application

- *Transport type:* Reliable or unreliable transport layer type (TCP/UDP).
- *Cycling port:* The transport port to use.
- *Local services:* This contains local configurations for using insert/extract modules. This field is optional.

Using the configuration set, the cycling application establishes incoming and outgoing links. Note that connection establishment is not required unless reliable transport is used. Using a make-before-break model (establish a new connection before removing old link), a new node can be inserted into an existing cycle without data loss. Simply put, each node ensures a receiving and sending links exist and are operational (by checking if the peer nodes are alive). This is a very basic and simple model, but it can be extended easily. Once the cycle module finishes this step, it is ready to handle incoming data. Nodes can also provide local services to other nodes to insert and extract required data.

To insert new data, the insert module establishes a connection with a client to receive data. It then reformats data and lets cycle module put it on the loop. One detail here is: *how do we identify data*? If the cycling nodes are the users of data, then there is no need for such identification (since they already know the content of each packet as they receive it). But, if an external node want to store and retrieve, then it would need to use a unique identifier for a data stream. The problem can be solved by generating a unique idenfier on insert requests which are then communicated to the client. If clients (such as a remote IoT device) just insert data for others, then identifiers are not required either. For clarity, we will make the following assumptions:

- Only nodes on the loop use data on the loop.
- External nodes can insert new data (when allowed) into cycle by requesting nodes managing the loop.
- External nodes *do not* request to extract data from cycles.

With the above assumptions, no data identification tags are necessary. Data is used or purged based on its content. Nodes on the loop act as direct users of data and will have access to full access to data as they are received on incoming links.

4.1.2 Objectives

We set the below points as the objectives of this experimental looping setup:

1. Demonstrate transient data storage using cycling in computer networks.
2. Measure the amount of data cycled relative to bandwidth and RTT of links.
3. Measure loss of data in cycling (due to packet drop at switches, errors etc.).
4. Compare reliable vs unreliable transport for cycling.

The above measurements help find optimal usage, which is used later in an IoT application.

4.1.3 Experiments

4.1.3.1 Experiment 1: Simple looping

4.1.3.1.1 Description: This experiment acts as a basic trial to show that data cycling works. 3 nodes as described earlier are setup with data cycling application running on each. We use UDP based cycling to establish that data cycling works. A single packet of a variable size is created and injected into the cycle. The packet is simply forwarded from one node to the next and kept on the cycle.

4.1.3.1.2 Results: By observing and logging each time a node sees the cycled packet for packet sizes of 50, 150, 500 and 1400 bytes, we can measure the wait times (or inter-arrival times as in Eq. 2.3). Figure 4.4 shows the variation against packet size. The variation comes due to the change in transmission times of different packets. Larger packets have bigger transmission delays.

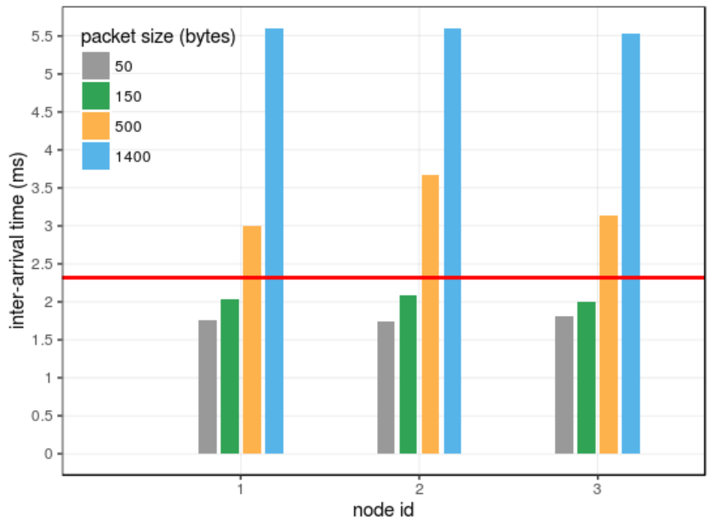


Figure 4.4: Wait time at nodes against packet sizes

The red horizontal line in Figure 4.4 shows the calculated delay of the loop as seen by measuring round trip time of each link and summing them up (and divide by 2). RTT measurement uses a 60 byte ICMP echo packet.

We can make an interesting traffic calculation using the above results. At node 3, when a 1400 byte packet is cycled, we fully receive it every 5.5 ms. The traffic flow rate is therefore 2.04 Mbits/sec. This isn't surprising when cycling is unregulated and is the traffic from 1 packet. Such traffic would degrade performance for regular traffic if not regulated. In fact the effect this would have was discussed in chapter 3 under statistical multiplexing. Such constant traffic would keep network links busy and cause traffic to lean towards a heavy tail distribution. The implication of this is: *data cycling needs timing regulation and cycling a lot of data will affect regular network performance.*

4.1.3.1.3 Summary: This experiment shows that data cycling is possible and measures wait times at nodes in the loop. It also discusses the negative effect of unregulated data cycling on network traffic and points to the need of regulation or delayed packet forwarding. Furthermore, cycling should restrict the amount of data being cycled.

4.1.3.2 Experiment 2: Overloading and packet loss

4.1.3.2.1 Description: In this experiment, multiple packets are cycled and the capacity of the cycling loop is tested. The objective of this experiment is to push the limits of the number of packets held in the loop. The methodology used is to start with an empty loop with no packets. One packet of a given size is injected into the loop every second (uniquely numbered). At each node we measure the number of unique packets seen in a period of 5 seconds. The expectation here is that all the cycled packets are seen by each node in every 5 second interval and is verified using packet numbers to be correct. The number of packets cycled as a function of time can be compared with injected packets to see losses and limits.

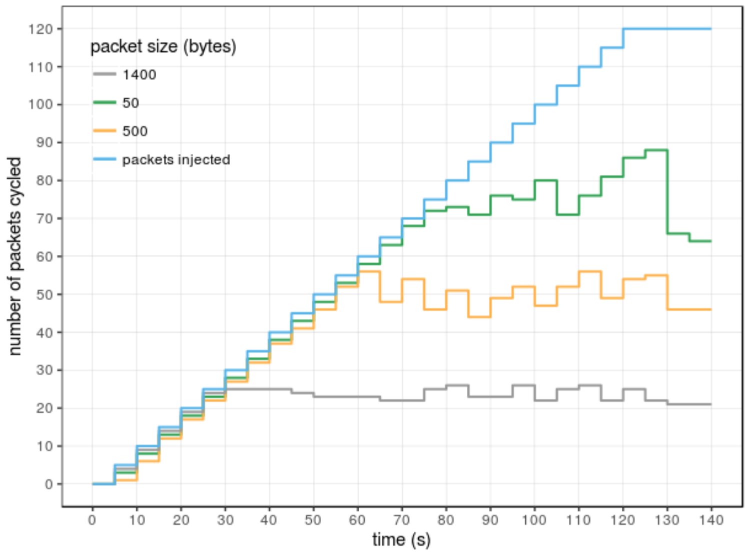


Figure 4.5: Packets cycled as more packets are injected with time

4.1.3.2.2 Results: Repeating the above experiment for packet sizes of 50, 500 and 1400 bytes, we can plot the number of unique packets seen by a node with time as shown Figure 4.5. It can be observed that for a given packet size, the number of unique packets follows the number of injected packets (blue) for a set time and then diverges and settles. After this point, all newly injected packets cause loss of some other packets indicating the upper limit of amount of cycled data. The limit is defined by the size of network buffers (aggregate) and is also depends on transmission delays corresponding to network links used.

It can also be seen that smaller the packets are, lesser the total of amount of aggregate data that is held on the cycle. For example, with 50 byte packets, settling is at about 70 packets which amounts to 3500 bytes. Whereas with 1400 byte packets, 20 packets are held in cycle, which is 28000 bytes. However, the percentage of packet loss follows a reverse trend with 41% and 83% loss for 50 bytes and 1400 bytes packets respectively. This is in order and is expected since UDP has no flow control and the packet losses are determined by the bottleneck network element or link. As in the previous experiment when a single packet was used, it was never lost (0% packet loss).

4.1.3.2.3 Summary: To summarize, it was seen in this experiment that cycling has a limit determined by transport mode, packet size and obvious limitations of bottleneck links in network. BDP is similarly related to bottleneck links. Smaller packets experience low packet losses and this corresponds to the properties of underlying transport protocol. Once, the limit is reached, packet losses stabilize keeping a constant number of packets on cycle.

4.1.3.3 Experiment 3: UDP vs. TCP

4.1.3.3.1 Description: Packet losses are common when an unreliable transport protocol such as UDP is used. In this experiment, reliable transport is used and is compared with UDP. The setting is different from the above. Here we inject 120, 500 byte packets back to back as TCP streams and measure the number of unique packets seen in 5 second interval and also accumulate uniques over time.

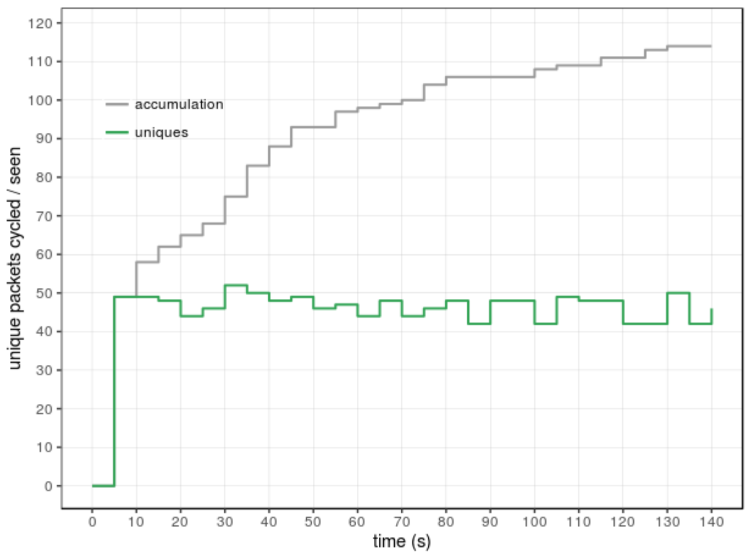


Figure 4.6: Uniques over 5 sec intervals and accumulation of uniques in TCP

4.1.3.3.2 Results: TCP provides acknowledged transport, which means packets aren't really lost. If lost, they are retransmitted. Figure 4.6 shows the number of uniquely numbered packets seen (green). Observing the difference with UDP is easy, since the number of packets seen in 5 second intervals is relatively constant since the start. This is due to flow control of TCP which adapts to losses. However, unique counting can be misleading since nothing is really lost in TCP streams. This can be seen by the accumulation line (grey) which simply computes a union of unique packets with time. All packets exist on the cycle, but instead in the TCP buffers of the 3 nodes. Unlike UDP, TCP buffers hold packets until correctly sent hence causing nodes to accumulate data.

4.1.3.3.3 Summary: It can be clearly seen that the idea of having transient data *in the network* is lost when using TCP since packets buffer up at nodes and the amount of cycled data is restricted by TCP window sizes (buffers). Due to such buffering, large wait times can be experienced before a packet is seen. It is easier to deal with lost packets than to wait a long time for packets to arrive. Hence TCP is not very useful for data cycling where we intend to use network delays for transient storage. But, small amounts (much less than buffer sizes) of data can be cycled on TCP links where reliability is required.

4.2 Smart Devices and Transient Data

The experiments in the previous sections bring out some characteristics of transient data and data cycling in wired networks. We observed that cycling of a large amount of data or even small amounts for a long time can produce unnecessary traffic which can clog the network, or in other words affect throughput of regular traffic on networks. Moreover, users of such transient data need to be loss tolerant, i.e, loss of some data should not affect their functionality drastically.

Given these observations, a class of applications in such a source-subscriber paradigm appears viable to make use of transient storage. With the advent and proliferation of "smart" connected devices (termed internet-of-things (IoT)), many new ways of data collection and access have been introduced. One of the most common ways is "cloud" storage and access. In such a mechanism, the "smart" devices generate data (say periodic windspeed measurement) and log them to a central cloud server stores data from where data can be accessed. Data access can either follow a request-response or a subscribe-notify model. In request-response, nodes send a specific content request which is processed and the response contains the data requested. In subscribe-notify, nodes subscribe to updated content and are notified with new data when it becomes available. In both cases, when a lot of nodes try to access common data (for example, weather), multiple responses/notifications are generated and directed to different nodes. Concretly, one message is sent to every node.

If we have multiple sources generating data that is accessed by many subscribers, the number of messages exchanged multiplies. Furthermore, if information needs aggregation by users using this data from different sources (like aggregate weather information from temperature, pressure, wind data from many nearby locations), then each endpoint has to do it separately. Although, this can still be done at a central location and distributed, the criteria of aggregation can be different for different groups of subscribers. Moreover, if the frequency of data generation is high, occasional loss of measurements during transmission should not affect functionality.

Accesses such as those described above can be simplified and made more efficient by forming loops of subscribers. In the example shown in Figure 4.7, there are 3 data sources of data S1, S2 and S3 and 5 subscribers which are looped to cycle data to exchange data generated by the sources. Nodes on loop cycle data by forwarding received data to the next connected node and this process continues to make information available to all subscribers. In fact, some nodes or node groups can aggregate data and forward that instead of same data received from sources, allowing context creation and maintenance in transient data. (Context here refers to aggregated or processed data).

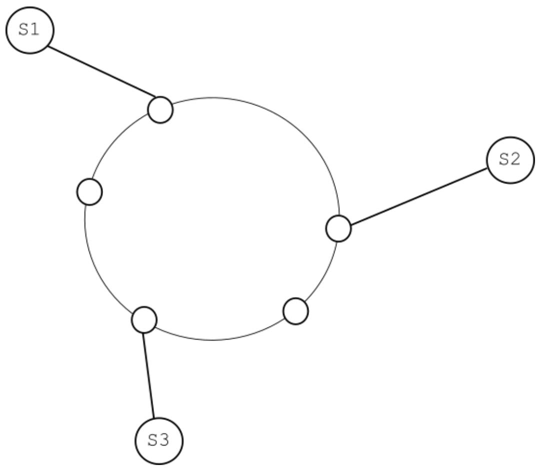


Figure 4.7: Connected devices and their subscribers

As we noted earlier, continuous cycling can affect regular traffic. However, in a cycle where data validity is ephemeral, nodes in the loop can purge data when it becomes invalid. The reason for invalidation can be many (Eg. validity expired, already processed etc.). This prevents cycling of invalid data. There can be many modifications done to the basic cycle shown here. As a concrete example, if groups of nodes subscribe to different sources, overlapping group cycles and subcycles can be formed to optimize node placement to change what data a node sees.

The above cycling method can have a significant advantages over conventional methods of data access by preventing central server overloading and preventing multitude of duplicated requests on longhauls. Furthermore, the need for central "cloud" is obviated, since sources can directly push data to required subscriber loops. This results in a distributed transient storage until such ephemeral data is consumed.

In the next chapter, an application of such transient context maintenance is presented in wireless nodes having ad-hoc networking capability.

4.3 Summary

In this chapter, a loop of nodes was setup to demonstrate the viability of data cycling in wired networks. Data cycling experiments were presented along with discussions on the effect of cycling on wired networks' traffic. Practicality of such loops was also discussed along with an idea of using data cycles in connected devices.

The next chapter explores data cycling in wireless networks and presents a real world application of cycling transient data.

Chapter 5

Cycling in Wireless Networks

Packet cycling in wireless networks can be very different from wired networks. To clearly differentiate, all networks using tangible physical medium (like cables, fibers etc.) are wired networks. Wireless networks contrarily use free propagation in vacuum or air as the medium for transmission and reception.

Wireless networks are complicated simply because of all the challenges a wireless network has to solve to communicate over a *shared common* medium in comparison to wired networks. Wired networks use shielded waveguides or electrical connections which provide a good level of isolation from the harsh surrounding environment. In case of wireless networks, the networking system should actively detect and adapt itself to handle changing environments. The mobility that wireless systems provide come at the cost of *complexity*.

The general challenges that wireless networks face in comparison to wired networks can be classified as below:

- *Interference:* Radio waves are affected severely by other electromagnetic (EM) radiation in their surroundings. Interference adds noise to signals making detection hard and sometimes changes the signal properties so much that detection in noisy conditions becomes impossible. Although various forms of modulation schemes have been invented to mitigate this, noisy environments remain challenging.
- *Channel sharing:* On wired networks, sharing can be implemented by detecting energy levels in the medium to check if it is being used. This is a hard problem in

wireless networks because channel sensing for small energy level changes during collision is simply not possible. Hidden node problem caused by completely undetectable nodes which can be detected by an access point for which both nodes are competing is challenging too. CSMA/CA is one mechanism used to solve this problem but it introduces latencies.

- *Propagation loss:* Electromagnetic radiation is subject to absorption, reflection , shadowing due to which power levels of signals fall rapidly with distance from the transmitter. Unlike wired networks where selective repeaters can be used, using repeaters can cause interference with the original signal and throughput drops by 50% for every repeater used.
- *Multipath effects:* Multiple paths taken by EM signals due to reflections from obstacles in their path can destructively interfere to cause weak spots. Furthermore, multipath signals arriving at a receiver can confuse it as the receiver has no way of knowing which one is more useful.
- *Dynamic environment:* Wireless devices have the freedom to move and this means the system has to adapt rapidly to changes. Timing offsets, frequency drifts and transmit/receive power change due to mobility and have to be detected and managed.

For the above reasons, wireless networks are designed to operate in ways that are different from wired networks. Hence, we need to revisit the effects of data cycling or looping in wireless networks from a different perspective. The next section analyzes cycling in wireless networks.

5.1 Cycling in wireless ad-hoc networks

Bandwidth-delay product in wired networks was simple to compute by multiplying end-to-end available bandwidth with round trip time. This indicates how much *in-flight* data there can be on a network link. It was also seen that higher this is, higher is the cycling capacity and transient data on that network link. In wireless networks however, the mechanism of operation changes the scenario. Wireless nodes need to contend for the channel before being able to send data. Moreover, only one node is allowed to send at a time in ad-hoc networks.

MACs of wireless nodes, contend for channel with backoff mechanisms and transmit when they acquire the channel. Next, they need to contend again to send the next segment of data. In a lot of 802.11 type networks, nodes have to also wait for ACKs from receivers before being allowed to send again. A clear contrast can be seen with wired networks where nodes sent data without waiting for sent data to reach its destination (like pushing packets into a pipe). A wired line can hold multiple packets back to back due to associated tranmission and propagation delays. However in mobile ad-hoc networks, due to stop-and-wait arrangement, only one packet can be sent at a time. If a sender tries to send multiple packets back to back, it will completely deny channel access to all other contending nodes for a long period of time. In fact some previous studies do show [5] that BDP cannot exceed the product of packet size (S) and round-trip hops (N). N here is the sum of nodes on forward (n) and return paths (m).

$$BDP = S \times (n + m) = S \times N \tag{5.1}$$

Note that enhancements like 802.11e use *burst-ACKs* to improve this slightly, but, due to really short propagation delays this can essentially be considered a single transmission.

How does this impact transient data cycling? So, if data cycling or looping has to be accomplished on a mobile ad-hoc network (typical setting for wireless sensor networks etc.) each node will need to *hold at least one packet temporarily.* If an ad-hoc network has N nodes then typically a cycle can hold N packets. If more needs to be cycled, then one or more nodes will need to *expand* temporary storage. Clearly, it can be very inefficient to try to “store” data on such a system using transience. However, transience here can be used in different paradigm, say to accomplish a co-ordinated task which would otherwise require centralized control. Data cycling can be used in modify-and-forward method to accomplish distributed tasks. Modify-and -forward here means that every node will contribute or change the incoming transient data before sending it out.

In the next section, vehicular networks (specialized mobile ad-hoc networks for automotive applications) is discussed and later, an application of data cycling in such networks is presented.

5.2 Vehicular networks

Mobile ad-hoc networks and vehicular ad-hoc networks have been studied for more than two decades now. What makes them so interesting is the dynamic and distributed nature of end points (nodes) that precludes central synchronized control. Furthermore, no network elements (routers, switches) are feasible in such networks. A good discussion can be found here [34]. This section introduces vehicular networks with some details of latest standardization schemes. Although detailed study is out of scope, a fair knowledge of network operation will be used in subsequent sections for developing simulations.

Vehicular networks were conceived with safety, congestion prediction, congestion avoidance, pollution control, electronic payment and infotainment in mind. The idea is to enable communication between vehicles (V2V) and between vehicles and infrastructure (V2I) to exchange information to achieve the above goals. For example, on discovery of an accident, vehicles moving towards the spot can be warned and instructed to take a detour to avoid a potential jam. For such interactions, wireless communication with these qualities is essential:

- A system of reference that ad-hoc nodes can align to for effective communication.
- Longer range (upto 1000 meters) of wireless communication.
- Ability to handle communication while nodes move at high speeds.
- High quality of service (QOS) in a dense ad-hoc wireless environment (many nodes).
- Good multipath resilience.
- Support different application classes (Eg. Emergency, Advertisement etc.).

IEEE formed the 802.11p standardization group after FCC allocated 75 MHz bandwidth (5.85 - 5.925 GHz) for dedicated short range communication (DSRC, a precursor meant for automotive communication). While 802.11p work group covered lower layers, IEEE 1609 work group standardized upper layers. Together 802.11p and 1609.x form wireless access for vehicular environments (WAVE) standard. Figure 5.1 shows the protocol stack.

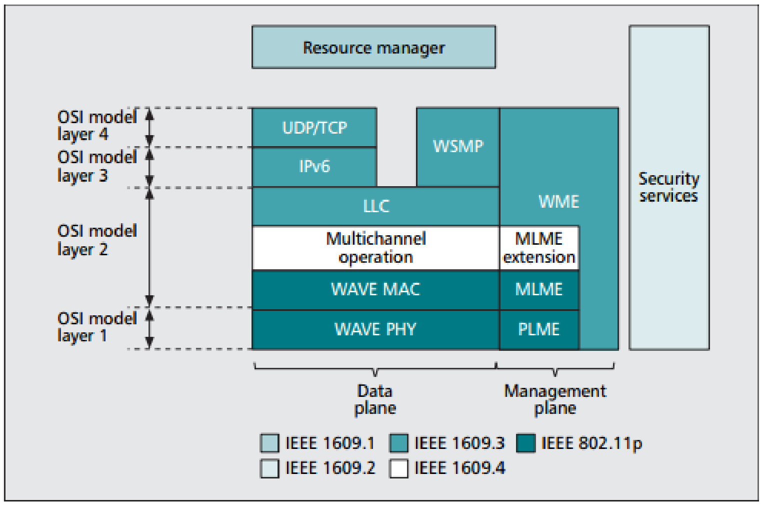
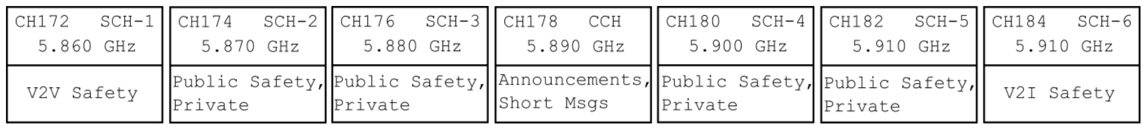


Figure 5.1: WAVE protocol stack [34]

WAVE defines two interoperating devices. Road side units (RSU), meant for vehicle to infrastructure communication with on-board units (OBU) installed on vehicles. OBUs enable vehicle to vehicle (V2V) data exchange. All WAVE nodes divide their time between two logical channels: control channel (CCH) and service channel (SCH). CCH is used for short messages, emergency services and announcements while SCH is primarily used for data exchanges. Figure 5.2 shows these time shared channels and frequencies. Every WAVE node synchonizes to CCH at the start of every UTC second. Timing is derived from an on-board GPS unit that WAVE mandates. GPS time is used as the reference for channel co-ordination. Nodes tune to CCH every 100 ms (for 50 ms duration). Nodes switch optionally to one of the SCHs during the subsequent 50 ms. This way there is always a guarantee that nodes tune to CCH at known times.

Message exchanges on WAVE are of two types: announcement frames and data



CH172 SCH-1 5.860 GHz	CH174 SCH-2 5.870 GHz	CH176 SCH-3 5.880 GHz	CH178 CCH 5.890 GHz	CH180 SCH-4 5.900 GHz	CH182 SCH-5 5.910 GHz	CH184 SCH-6 5.910 GHz
V2V Safety	Public Safety, Private	Public Safety, Private	Announcements, Short Msgs	Public Safety, Private	Public Safety, Private	V2I Safety

Figure 5.2: WAVE channels

frames. Announcement frames provide WAVE basic service set (WBSS), configuration information, to groups for using SCHs. These are sent on CCH. Data frames are used for data exchange on both CCH and SCH. Specifically two kinds of messages exchanges are possible on WAVE. First is wave short message (WSM, using WSMP) . These are CCH or SCH messages which use a broadcast communication scheme. Second is IPv6 messages , which are sent only on SCH and can provide IP services. WAVE nodes have to join some available WBSS to use IP services. WSMP messages typically are on control channel and utilize the same 802.2 LLC services as IPv6, but its addressing scheme uses MAC addresses instead of a network layer address like IPv6.

Channel access at the physical layer is done using standard channel sense multiple access with collision avoidance (CSMA/CA) with EDCA extensions which are well known 802.11 networks. A good overview with specific details of WAVE operation can be found in [34]. The basic information introduced in this section is sufficient to discuss a cycling algorithm in dynamically changing network such as vehicular ad-hoc networks (VANET).

5.3 Cycling in V2V networks

In this section, a novel application of cycling in vehicular networks is presented in a fast changing vehicular environment. As discussed previously, such dynamic networking environments present difficulties for centralized or even de-centralized but coordinated control. In the following sections, a simple problem involving autonomous counting of vehicles in a closed boundary is discussed. Counting vehicles (or estimating vehicular density) can be particularly useful in traffic assessment and management. Infrastructure necessity, well co-ordinated traffic flow, peak hours' prediction, traffic prediction can be done very effectively by knowing the density and flow of vehicles through geographical regions. This can include populated urban areas, suburban intersections or even highway sections. Counting vehicles has been a prevailing problem and it has always been expensive to install cameras or loop detectors to estimate traffic.

Here we use the futuristic WAVE (already commercial) based vehicles to estimate counts without any assistance of external infrastructure. A part of motivation for counting apart from above mentioned uses, is also the rise in autonomous vehicles on road. If these autonomous vehicles can get data feeds from analytics using vehicular density (derived from counting), they will be able to self manage routes or make even complicated decisions like where to park or how long to wait to avoid jams for faster commutes. In fact with the rise of self driving cars, local accumulative counting of vehicles in surroundings can aid autonomous vehicle directly since they do not have to rely on any central control for traffic assessment. Another use of counting is frame and channel management in WAVE which, studies have shown, need to be varied according to local traffic density. The procedure is presented in parts involving methodologies, simulations and results.

5.3.1 Problem and assumptions

To formally state the problem: we are trying to count the number of vehicles that enter a particular grid (defined by geographical boundaries) in a given period of time without using any external equipment assistance or triggers. No roadside units (RSU) are employed in the procedure. However they may be later used for data forwarding. This is a tricky problem since each node cannot individually count by making a list of all nodes they detect. Although such a methodology seems to work at first, imagine what happens when a new node enters the grid and some of the earlier nodes have already left the grid. The new node will be unable to see previous context. So we need a different methodology which maintains the past context for new nodes.

We make the following general assumptions for the development of this application.

- Each vehicle is equipped with WAVE hardware and protocol stack allowing it to communicate using IEEE 802.11p - 2010 standard of WAVE with other vehicles and roadside units equipped with the same.

- Each vehicle has an onboard global positioning system (GPS). GPS is used and mandated by WAVE. Hence it is a requirement for WAVE hardware and protocols to function correctly. The system is used for two reasons. First, it allows for time synchronization (used by WAVE) and second, it enables precise localization, allowing vehicles to know when it is in or out of a certain region (grid).

5.3.2 Counting algorithm

As discussed previously, WAVE nodes (vehicles) are allowed to communicate on upto 7 channels on different frequencies. The decision of which channel to transmit on or to tune for reception is made by processing advertisements or schedules sent on the control channel (CCH), CH172. A node tunes to CCH every 100 ms, and is required to remain on that channel for 50 ms, before it can switch to one of the other service channels (SCH).

Nodes can decide to not switch to SCHs at all. In this application we will use only CCH for all tranmission and reception. Introducing channel switching is trivial and will affect the application minimally.

5.3.2.1 Algorithm

The algorithm describes autonomous accumulation (counting sum) based on modified data cycling. The term *modified* is used here to emphasize that every node modifies packets slightly before retransmitting them. We will use a running example to illustrate the algorithm used for packet cycled counting and its associated intricacies. The algorithm for counting is described stepwise below. To enable such counting, we define a *counting packet*, which is a message exchanged between peer counting applications. This packet can be encapsulated in a WAVE short message protocol (WSMP) packet or an IP packet. A counting packets contain the following fields.

1. *Header:* Header to identify a "Counting Packet."
2. *Grid ID:* This field contains information of geographical boundaries where this counting packet is valid and nodes can use the packet for counting.

3. *Sequence:* This number uniquely identifies a time slot in which this counting packet is valid. For example, a day may be divided into 5 minute intervals uniquely identifying each one. This is possible since all nodes in WAVE have GPS reference time.

4. *Node list:* A list indicating which nodes have participated in counting so far using this packet. This can be a *Bloom Filter* containing hash marked bit fields or a list of medium access control (MAC) addresses. This is the accumulated list and forms the "context" of "state" stored.

5. *Count:* This field is optional and indicates the total count for quick reference. This number can be deduced from *Node list* field.

Step 1 *Initialization:* Applications at all nodes clear their packet caches and clear all variables of counting. All timers are stopped and reset. This is the starting condition of applications on all nodes. As an example, let's assume we have vehicles A, B, C and D initialized to *reset state.* A, B and C are reachable to each other (by wireless radio). Node D is reachable only to node C.

Step 2 *Trigger:* One of the vehicles in the grid triggers counting. Which node acts as the trigger is decision that can be randomized. Even with many triggers, the algorithm will work (in fact better) and this discussed later. Triggers occur in allowed *counting grids*, which are simply geographical boundaries used for decision making. Nodes inside the boundary are allowed to participate, and those outside are not. Moving out of a grid disallows a vehicle to participate in counting procedure even if it was previously allowed. Counting is triggered by simply transmitting a *counting packet* (identified by a specific application header) indicating the current grid, time sequence number and by marking itself in the node list. Let us assume vehicle *A* triggers counting. The packet is broadcasted as a WAVE CCH packet after a random time determined by a *wait timer (WTMR).* The duration of wait can range from a few microseconds to a few milliseconds. WTMR's randomized wait prevents channel crowding.

Step 3 *Broadcast reception:* Transmission happens by broadcast. The packet contents are now [Grid = x, Seq = 1, Count = 1, List = {A}]. All nodes that receive this packet check if they currently belong to the same grid as indicated by the counting packet and if the current time sequence number matches the one in the counting packet. If not, they simply discard the packet. This check is shown as *Can participate?* in Figure 5.3. Node B and C perform this check and pass it. Node D does not receive this broadcast.

Step 4 *Participation:* If a node passes "Can participate?" check, then it proceeds to check if it has already participated in counting as indicated by this packet. The node checks this by querying the included nodes list. This constitutes the *Did participate?* check. If it already has, then the node discards this packet. If not, then it adds itself to the nodes list, and increments the count by one if present. The modified packet is added to an internal transmission queue and WTMR is started to await transmission. In our example, B and C each prepare their packets. B's packet = [Grid = x, Seq = 1, Count = 2 List = {A,B}]. C's packet = [Grid = x, Seq = 1, Count = 2 List = {A,C}].

Step 5 *Wait timer expiry:* When WTMR expires, the node dequeues packet from the transmission queue and transmits it. Lower layer CSMA/CA still apply. In our example, C's timer expires first and it broadcasts its queued packet.

Step 6 *Repetition:* Steps 3, 4 and 5 are repeated as the counting progresses. WTMR restricts excessive broadcasting. Since node C transmitted its packet first, A, B and D receive it. Node A discards the packet since it sees that it has already participated. Node B, on reception of this new counting packet, trashes the previously enqueued packet. It then prepares a new packet whose contents are [Grid = x, Seq = 1, Count = 3 List = {A,C,B}] and enqueues. Node D does the same and its packet = [Grid = x, Seq = 1, Count = 3 List = {A,C,D}]. Plain counting would stop here. A unique converged count was not achieved since the bridge at C did not retransmit. Step 7 tries to mitigate this islanding problem.

Step 7 *Auto re-broadcast:* A secondary procedure is used to achieve count convergence during possible island-bridge problems (D is the isolated island, C is the bridge to the mainland with A and B). Every node temporarily holds on to the latest valid counting packet even if it has already participated. A re-broadcast timer (RBTMR) is started in background when participation is allowed and transmit queues start idling. RBTMR is a much longer timer and runs in the range of several seconds. Upon expiry and persistent inactivity nodes enqueue the "latest" packet and start WTMR for transmission. This forms a slow outer loop to achieve convergence. In our example, auto re-broadcast causes A,B and D to eventually count one another. The final packet may look like [Grid = x, Seq = 1, Count = 4 List = {A,C,D,B}].

Step 8 *Termination:* Counting terminates when time progresses beyond the valid sequence or nodes exit the allowed counting grid. They may restart counting again in another grid starting back from Step 3, only this time resetting all variables as in Step 1 because the grid is different.

This completes the counting procedure. Every node that has participated in counting hold a count from the counting process. This count is updated in every vehicle that receives it and delivered later to a roadside unit or a central server directly. Note the subtle assumption that the MAC address node lists or the Bloom Filter hashed bits uniquely identify a vehicle during the period of counting. This should be safe since address change happens only once every 5 minutes or so. Collisions in Bloom Filter might occur and there can be false positives during look up, but excellent performances can be achieved with relatively short bloom filter sizes.

The pictographical control flow for the above described counting procedure is shown in Figure 5.3.

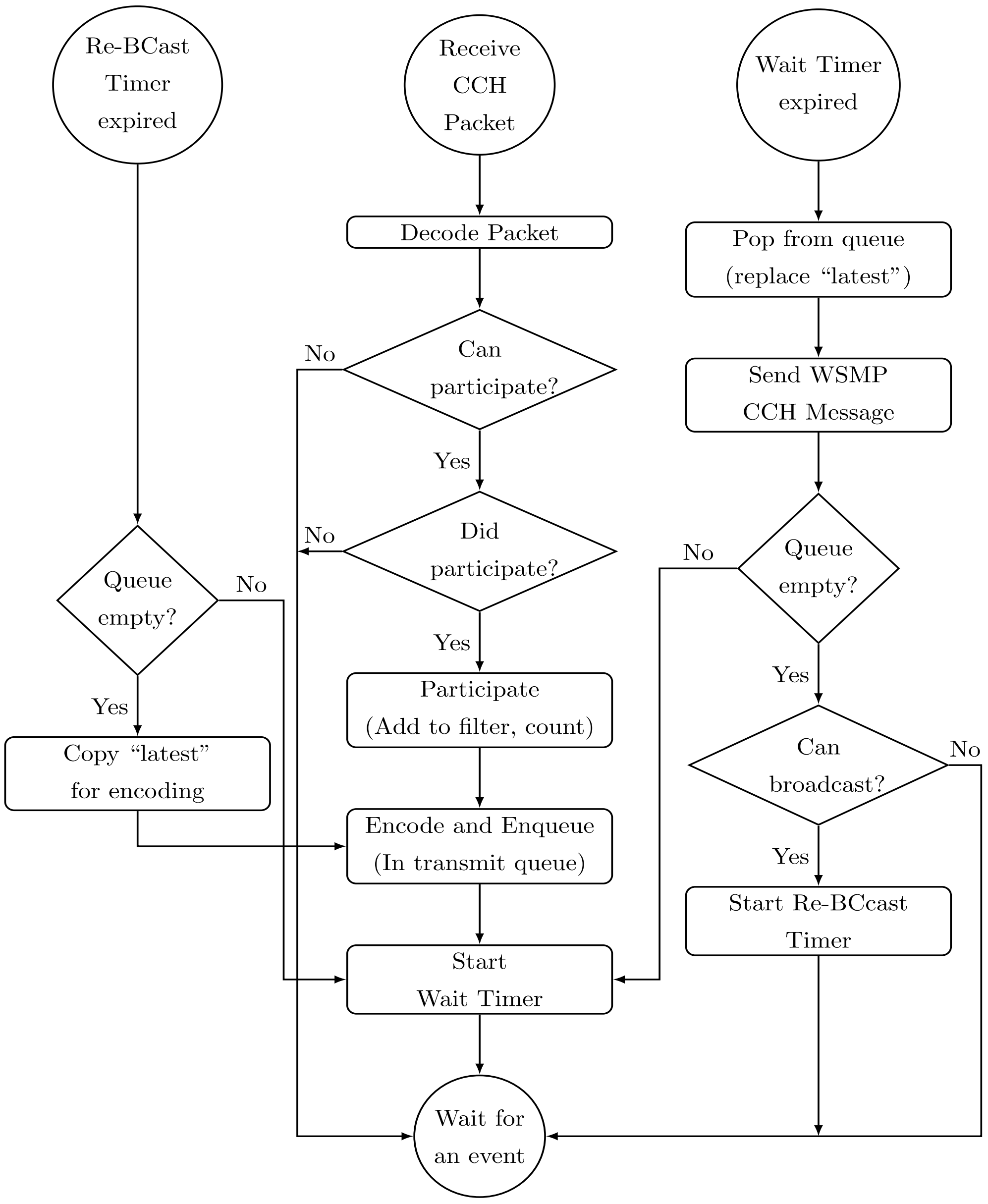


Figure 5.3: Counting control flowchart

5.3.2.2 Analysis of the Algorithm

Although mobility of nodes was not considered in the above algorithm description, the procedure does not change when the nodes are in motion. Transmission ranges play an important role when nodes are mobile. However, it is easy to see the complexity of central control in such dynamic environments. This is where packet cycling solves the problem of distributed counting by constantly updating nodes' "latest" count. By constantly resending packets to one another, the nodes in the grid *update and store* information giving an effect of transience.

To understand this a little better, consider a large area with the bounded rectangle representing the allowed counting grid, shown in Figure 5.4. The nodes with red circles are allowed to participate at time "s". The nodes are in constant motion so they can exit the boundary conditions at any time. Once they do, they stop counting (but take away the last updated count with them). These nodes are shown in Figure 5.5 with green circles. At time s, the nodes would have reached an agreeable count, say *k*. This k is now temporarily *stored* in the grid by counting participants.

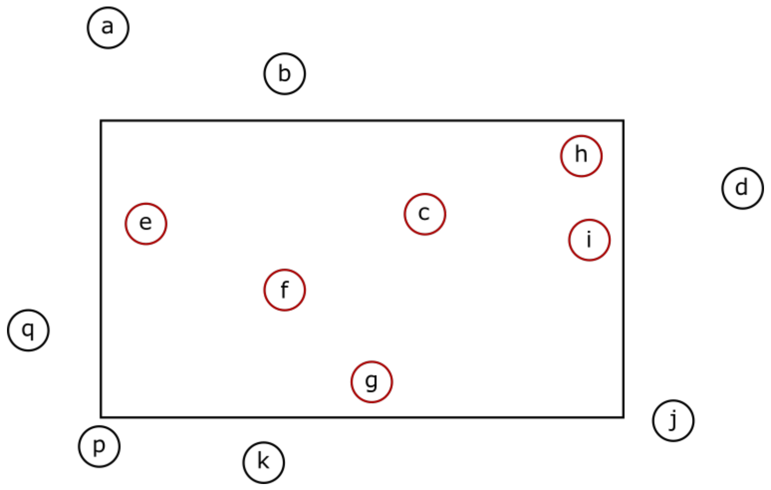


Figure 5.4: Nodes at time "s"

Next, due to movement the situation changes to Figure 5.5. New nodes now qualify to participate in counting (Eg. Nodes k, d and q). Using the broadcasts they receive from nodes that have already participated, they can add themselves to received nodes' list. This gives a cumulative count of the number of nodes that have entered this grid. The cumulative memory is self maintained by participating nodes and passed over to new arrivals. This way, old nodes' list is extended by newer nodes. Hence the count represents the number of nodes that have *entered* the grid during that time period (indicated by sequence number in packets). The node flow rate can be easily determined using this information. Note that both s and t lie inside one time slot as determined by sequence number.

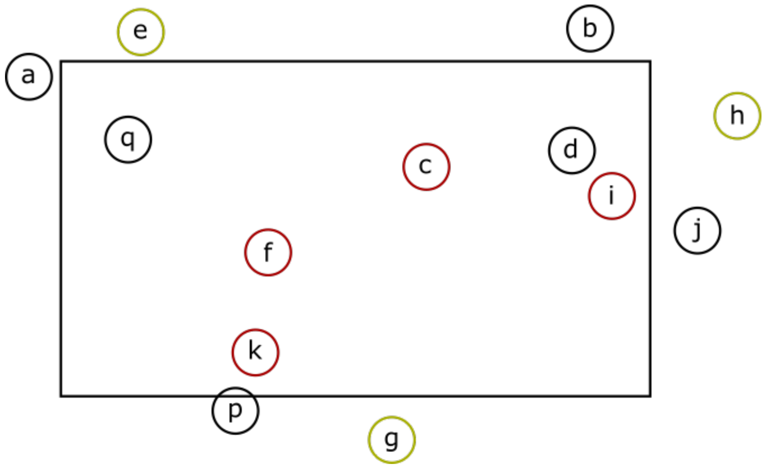


Figure 5.5: Nodes at time “t”

Counting is terminated and a new accumulative counting is started when time boundaries are crossed. Older counting packets lose their validity are hence discarded. Note however that the nodes which have moved out would have already taken most recent copies with them. The following points summarize the main observations of the above algorithm:

- Temporal transience is created by nodes resending counting packets to each other, randomly but persistently. This creates accumulative storage effect inside the grid.
- If all nodes within a grid are reachable to one another then N broadcast transmissions are sufficient to obtain a converged count. This the best case, provided no two wait timers expire simultaneously and new broadcasts are processed before expiry.

- The amount of storage required at each node for the base algorithm (without auto re- broadcast) is zero. If auto re-broacast augmentation is used, then one counting packet should be temporarily held by every participating node.
- Although CSMA/CA of 802.11p MAC self limits collision and exponential packet exchanges when the nodes run freely, an application level timer (such as WTMR) is required to effectively control broadcasts to prevent packet drops due to very closely spaced broadcasts.
- This procedure can be run as a WAVE application on a public safety application channel such as CH172, with a lower priority than advertisements and emergency broadcasts.

5.3.2.3 Islanding Problem

The problem of *node islands* was shortly discussed during algorithm description. Formally, the problem is that: there can be multiple nodes or node groups which are either sparingly reachable or not inter-reachable. Each of these groups can hence end up with a local count instead of a global cumulative count. In the running example, node D and group {A,B,C} formed two islands but was bridged by node C. Without any augmentation beyond Step 6 of the algorithm, node D would have ended with with a count of 2 (C and itself) and A,B would have never learnt of D's presence.

To mitigate this problem, Step 7, auto re-broadcast mechanism was introduced. In auto re-broadcast nodes keep a copy of the most recently received valid packet. When inactivity is sensed, a secondary timer (auto re-broadcast timer) is started with a random time ranging from a few 100 milliseconds to several seconds. Inactivity at nodes can be sensed in several ways. One easy yet effective way of doing it is to check broadcast transmission queue status. Queued up packets indicate activity. Another way of detecting inactivity is to run timers indicating the time since last broadcast. Once, auto re-broadcast timer expires, the most recent copy ("latest") is queued for broadcast and is transmitted with the same wait timer mechanism. Auto re-broadcast has the effect of *updating* isolated island counts in case there are bridges by retriggering localized counting in nodes or groups missing in island counts and vice-versa for islands. However, if there are truely isolated islands with no bridges, it has no effect unless a bridge is formed later due to movement. In situations of varying speeds and intersections, which are common in vehicular transport, bridges can be often expected to form.

5.3.2.4 Bloom Filters vs Node Lists

Nodes' list was an integral part of counting packet. Nodes' list enables nodes to participate and prevent duplicate participation by check-before-do method. The node list can be formed in two ways. The first straightforward way is to add a node's identity into the list directly. This identity can be an IP address, MAC address or an equivalent form of unique address. Inclusion of MAC address involves storing 6 bytes of data to identify each node. As the number of participating nodes increases, the list grows linearly ($O(n)$) and so does the size of the counting packet. For vehicular networks it's a common practice to keep packet sizes small ($<$ 1024 bytes) [28, 29] to avoid excessive packet loss rates and to increase packet delivery probability over a longer ranges. Moreover, the maximum allowed size on WSMP is 1400 bytes. Using MAC address lists allows participation of around 200 nodes before sizes exceed the statute maximum. More over as sizes increase, the delivery chances of CCH broadcast falls quickly. Lists can be hence inefficient in such environments. Lists also identify MAC addresses resulting in privacy concerns (although WAVE nodes randomize MAC addresses every 5 minutes). Furthermore, searching long address lists is time consuming. Such a search generally takes as much time as the length of the list. Hence, the time complexity of this search is $O(n)$.

The requirement here is for nodes to able to just detect previous participation. Hence we need a mechanism to efficiently store this information. *Bloom Filters* [3] provide a perfect solution to this problem.

Bloom filters are bitfield data structures which have m bit bitvector and k hash functions which return a decimal value between 0 and m. Every element (or key) that needs to be inserted into a bloom filter, is run through all k hash functions. The bits corresponding to the outputs of hashing are set in the m-bit vector to mark the presence of that key. To look up the presence of a key, k functions' hashing is repeated with that key and the corresponding bits are checked in the vector. If all of them are set, then there it is *possible* that the key is present. If any of the bits are found to be 0s, the the key is *definitely* not present. Since a bit can represent multiple keys in aggregate (after many insertions), there are chances of false positives. Typically, good hash functions with a long enough bloom filter can attain a false positive rate below 1%. Bloom filters never return false negatives, that is, they will always return true for keys that have been inserted. Once inserted, a key cannot be deleted from regular bloom filters. This is because any given bit in the n-bit vector can represent multiple keys. Resetting a particular bit in the bit vector affects multiple inserted keys. There have been modifications to bloom filters to allow deletions. These are called counting bloom filters. However, this property is not useful for counting where basic bloom filters are sufficient.

There are many advantages of using bloom filters. The most significant one is the constant size used to represent many keys. Counting packets can be of constant size even when many nodes are counted in. The size of a bloom filter is typically defined as the size of the m-bit vector associated with it. For bit vectors of size 512 bytes (4096 bits), with about 6 hash functions can achieve a false positive rate of less than 1% for upto 500 nodes! So, bloom filters provide significant space advantages. Another main advantage is the search time. Any given key can be searched in a bloom filter in $O(k)$ time, where k is the number of hash functions associated with the bloom filter. Since k is typically small (less than 10), search time effectively becomes a constant and does not scale with the number of participating nodes.

As mentioned earlier, there are some compromises or tolerances required to use bloom filters. First, and the biggest is the possibility of false positives. Although low rates can be achieved, there is always a finite possibility that a key that was not inserted into a bloom filter is identified as inserted. In other words, nodes detect participation although not having done so. So, applications have to find other ways to achieve deterministic results. Next, since a key cannot be removed once inserted, set operations which involve deletions will face difficulties operating with a bloom filters unless augmented or modified [14].

In the next section this algorithm is implemented and evaluated for various scenarios in vehicular networks in a simulator (NS-3).

5.4 Simulations

Simulating new algorithms and methodologies is a standard way of verifying their effectiveness in *close to* real environments. Most aspects of real wireless communication problems have been modeled in various simulators. Network simulators specialize in providing vast scientific and research based models for networking and communication problems. *NS-3* is one of the most used and verified network simulators [8]. The counting algorithm is implemented as an application over the 802.11p WAVE protocol stack implementation of NS-3.

5.4.1 Models and Implementation

NS-3 uses encapsulated objects to model hardware, software and medium of communication. Figure 5.6 shows NS-3 nodes. Each node can have multiple network devices which are abstract network interfaces (like Ethernet, Wi-Fi, WAVE etc.). The network device is accessed by applications through protocol stacks allowed on that network device. Components of protocol stack can be assembled by choosing each layer and configuring it.

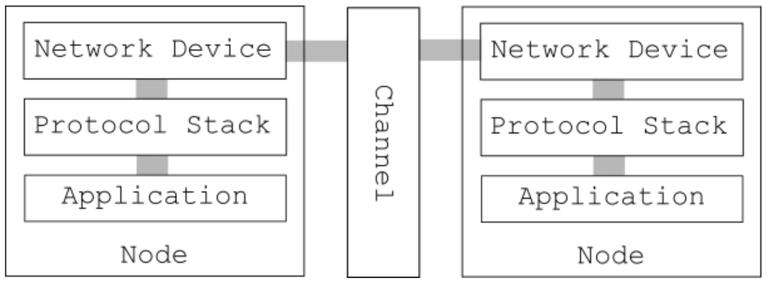


Figure 5.6: Two nodes in NS-3 communicating over a channel through a network device

For simulating applications on vehicular networks, the following models will be used. The choice of these models is determined mostly by IEEE 802.11p standard for vehicular ad-hoc networks. However, some channel specific radio frequency fading and propagation models are chosen with acceptable trade-offs between accuracy and simplicity. Note that most of these models can be easily changed to select different ones. For the purpose of the rest of this simulation we will use the below models and configuration unless otherwise specified explicitly.

1. *Application layer (APP):* The counting application is layered on the WAVE protocol stack, using WSMP CCH as its data plane. Currently there is some standardization done through SAE J2735, IEEE 1609.1 following ASN.1 message encoding rules. For simplicity, here we won't use any application standards. Conforming to these standards is a matter of changing message formats and priorities. The counting application uses a simple counting packet as defined earlier with octet aligned bitstream encoding.

2. *Network layer (NET):* WAVE uses two possible methods for transport and network layer services. WSMP acts as both transport and network layer service for short broadcast (CCH) and services (SCH). The other possibility is to use TCP/UDP with IPv6 for generic data exchange applications on SCH. Counting application is implemented using WSMP over CCH and later channel switching is discussed. Moving from WSMP to UDP/IPv6 involves adapting node identification to IPv6 instead of WSMP. These layers are defined in IEEE 1609.3 standard and is implemented in NS-3 as *WaveNetDevice*.

3. *Data link and MAC layer (MAC):* WAVE uses 802.2 logical link control (LLC) standard and IEEE 802.11p for general MAC specification. LLC provides standard services as discussed previously. We use a no quality of service (nQOS) MAC model similar to WiFi in ad-hoc mode implemented in NS-3 as *OcbWifiMac*. Multichannel operation as defined in IEEE 1609.4 is also used. A predetermined EDCA set of configuration parameters is used for CCH WSMP operation.

4. *Physical layer (PHY):* The physical layer closely parallels other 802.11 standards such as WiFi and is defined by IEEE 802.11p standard. It is implemented in NS-3 in *YansWavePhyHelper*. At any time PHY can be in receive, transmit or sleep state. The energy detection (ED) threshold is set at -96 dBm. OFDM with 10 MHz bandwidth and 6 Mbits/sec is used as the modulation scheme (half rate QPSK).

5. *Channel (CH):* Standard wireless radio channel models are used as the communication medium. The channel uses an error rate model based on additive white

gaussian noise (AWGN) for OFDM modulation schemes [27, 7]. The channel also models delay in radio signals and the path loss they experience during propagation. The next two points define these important channel parameters.

6. *Path loss:* All radio signals attenuate during propagation through a medium due to absorption, diffraction, reflection, free-space loss etc. The propagation loss model tries to capture these effects to simulate more realistic scenarios. Vehicular networks are normally line-of-sight networks (not always true). So the Two-Ray Ground path loss [9] model has been used for most research in such networks. The loss in itself is given by the equation 5.2 where, P_t and P_r are transmit and receive powers (W), G_t and G_r are transmitter and receiver gains, H_t and H_r are antenna heights from ground, d is the distance between nodes and L is a fixed system loss.

$$P_r = \frac{P_t \times G_t \times G_r \times H_t^2 \times H_r^2}{d^4 \times L} \tag{5.2}$$

The most important result that comes out of this model is the upper limit on range of transmitted signals. With a maximum transmission power of 50 mW (17 dBm), the range is about 250 meters. Note that this is the distance where the signal power falls below -99 dBm and the receiver drops it. For shorter distances where Two-Ray Ground model, works incorrectly, Friis free-space path loss model [16] is used.

7. *Propagation delays:* Delays are experienced by radio waves simply due to distance propagated. The used delay model is a constant delay model where the delay is computed by dividing the distance propagated by the speed of light.

8. *Mobility:* An essential component of vehicular networks is mobility. A lot of design considerations in WAVE are because of mobility of nodes. In this simulation random wandering nodes (two dimensional) and linear moving nodes (one dimension) are used. The first approximates movement in general and the second tries to model highways. Experiment 3 is dedicated to the variations in mobility.

9. *Bloom Filter:* Bloom filters in counting packets uses a 4096-bit (m) bit vector with 6 hash functions (k). The primary hash function used here is Murmur Hash

3 [40] which produces a 128-bit hash. This hash is split to two primary hashes A and B of upper and lower 64-bits respectively. Using A and B, subsequent hashes are produced by double hashing [21] using the below formula in Eq. 5.3 where i varies from 0 to $k-1$.

$$H_i = A + i \times B \tag{5.3}$$

Such hashing results in fast performance without loss of statistical properties of bloom filters [21]. Number of keys (N_c, count) are estimated from bloom filter using the formula in Eq. 5.4, where X is the number of bits set to 1 in the bit vector.

$$N_c = \frac{-m}{k} \times ln[1 - \frac{X}{m}] \tag{5.4}$$

The details of this estimation can be found in [31].

5.4.2 Experiments and Results

The counting algorithm along with all the specified models was implemented in a comprehensive simulation module in NS-3. Each experiment presents a setting and discusses corresponding results.

5.4.2.1 Experiment 1: Simple counting

5.4.2.1.1 Setting: This is a basic experiment with the below settings with nodes placed randomly in a $5km \times 5km$ grid. We use a single node trigger and standard counting packets. Tranmission power is varied to observe time taken by nodes to converge to a count. Packet size growth when using nodes' list is measured and contrasted with bloom filter usage. Nodes do not move out of the grid in this experiment, so they are always qualified for counting. The wait timer period is drawn randomly in the range 0 – 100ms while auto-broadcast timer varies between 2 – 3 seconds.

5.4.2.1.2 Results: Figure 5.7 shows the convergence of counting with time. Nodes moving randomly at speeds between 0 and 20 m/s in 5km grid. Observing the counting variation with density of nodes, it is clear that as the density of nodes increases, the error between average count and real count increases and then decreases again. The increase is due to interference in the environment as density increases and the later decrease is due to rise in signal-to-noise ratio as nodes come closer. The count value growth is logarithmic with a large jump in the first few 100 milliseconds.

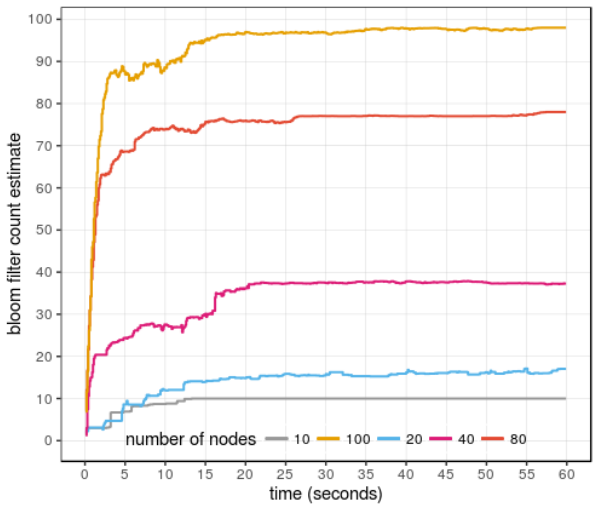


Figure 5.7: Average count at nodes in random motion

The aggregate number of packets sent (on an average) by node over time increases linearly with time. Figure 5.8 shows this. Note that this plot shows cumulative packets sent.

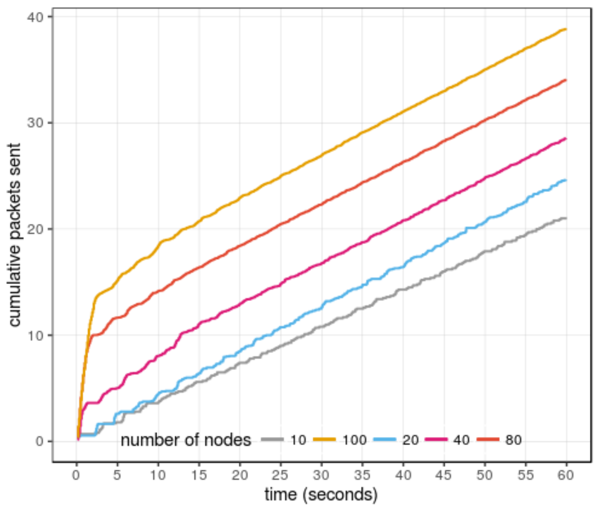


Figure 5.8: Average packets sent over time

Next, if we switch out bloom filter for nodes list (using MAC address as the identifier), then we can observe that packet size growth is logarithmic too as plotted in Figure 5.9. This implies that packet size grows linearly as counting progresses.

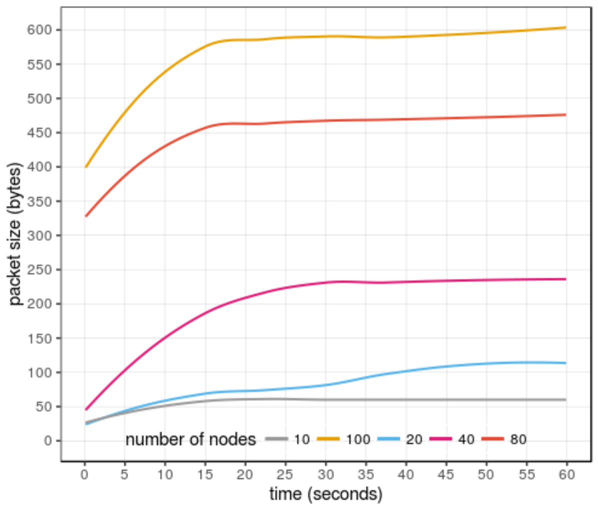


Figure 5.9: Change of packet size when using lists

5.4.2.1.3 Summary: Simple counting progresses rapidly on trigger and settles. Initially since all nodes need to participate, the number of packets broadcasted increases rapidly. But later when most nodes have participated, broadcast rate decreases and settles to become a constant. We also note how interference due to node density impacts counting.

5.4.2.2 Experiment 2: Effects of varying timers

5.4.2.2.1 Setting: In this experiment a $5km \times 5km$ grid is used, but with a fixed 20 nodes randomly placed. The nodes move with random velocities with speeds in $0 -$ 20 meters/sec range. One of the nodes triggers counting. No nodes leave the boundary during simulation (they rebound).

5.4.2.2.2 Results: To first analyze the effect of varying the wait time (WTMR), we use the same metric: time taken for node counts to converge to the correct value. By varying WTMR from really short times of the order of 5 ms to very large values comparable to auto-broadcast timer, we see the convergence as shown in Figure 5.10. The figure shows mean wait timer duration and the timer can vary ±50 ms about this mean.

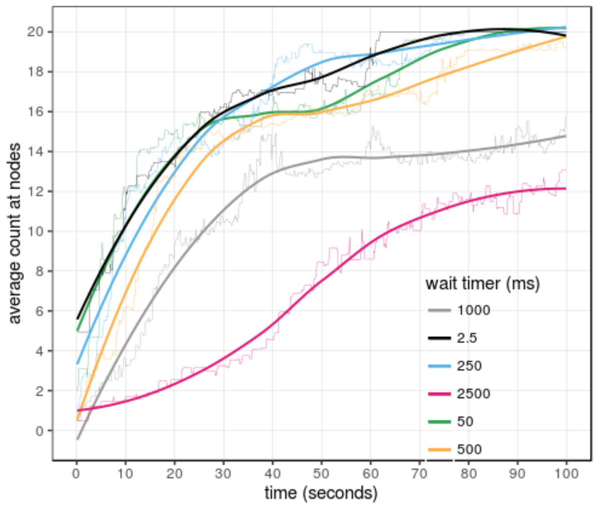


Figure 5.10: Count convergence with varying wait timer

Obviously, a large wait timer performs badly as expected. However, a really small wait timer performs badly too. This is because of large number of closely spaced broadcasts caused by a small timer. This aligns with observations of performance of packet delivery probability with changing contention window (CW) [12]. The best performance is seen with a 250 ms timer. Note that the corresponding sent and received packets change with wait timer as shown in 5.11.

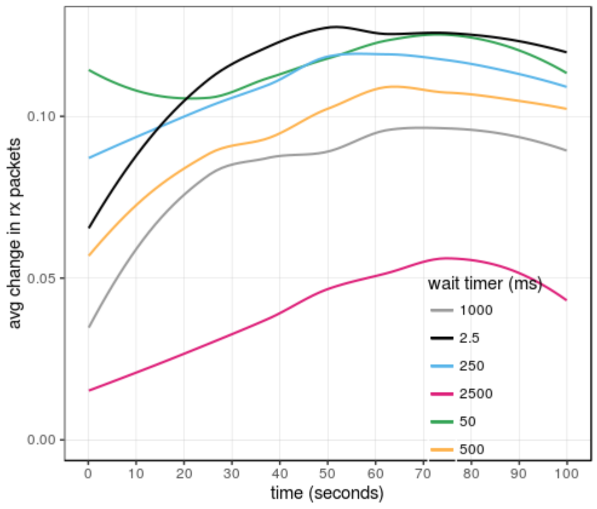


Figure 5.11: Average change in number of received packets at nodes

By setting the wait timer to vary between 200 to 300 ms, we now vary auto rebroadcast timer (ABTMR). Figure 5.12 shows the effects of this variation. It can be observed that larger ABTMRs cause oscillations in counting progress.

This is because, when many ABTMRs expire together, cached old counting packets are re-sent by nodes which would've have moved very far away from their earlier counting locality. In the new surrounding counting starts again with this unseen packet. To balance, an ABTMR duration whose mean is about 10 times that of wait timer performs well with no big dips and re-counting (Eg. 2500 ms).

5.4.2.2.3 Summary: It can be concluded from this experiment that WTMR and ABTMR have optimum values which can be chosen as discussed above. These timers can be expected to vary with sparsity and mobility of nodes as will be seen later.

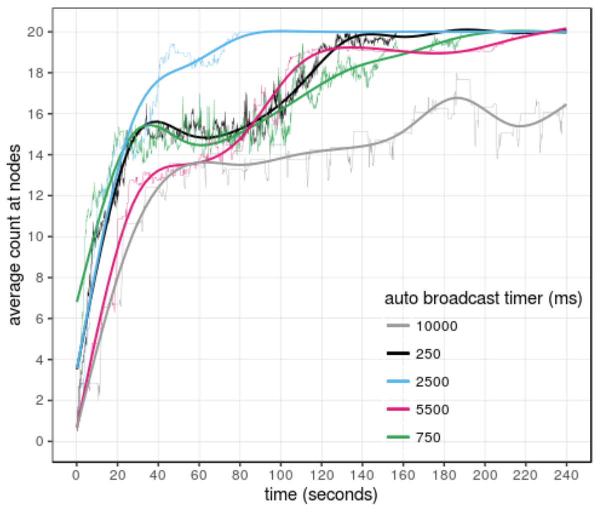


Figure 5.12: Count convergence with varying auto-broadcast timer

5.4.2.3 Experiment 3: Effects of mobility

5.4.2.3.1 Setting: To analyze the effect of mobility on counting, we use two distinct settings. First, as before, we use a random mobility model with varying node velocities. We use a $10km \times 10km$ area for node movement and use its innermost $5km \times 5km$ as the counting grid. This way, nodes are allowed to enter and leave the grid. Next, we simulate a linear section of a highway using motion only along X co-ordinate and shrink the counting grid to a rectangle covering a specific section of the highway. In both cases, upper limit on random speed is varied to get the nodes to move faster.

5.4.2.3.2 Results: Counting is a cumulative process as discussed before. To compare the impact of speed on counting process, we measure the number of unique nodes that enter the counting grid over a period time. Ideally, this cumulative number should be the count. We also measure the estimated count in all nodes inside the counting grid.

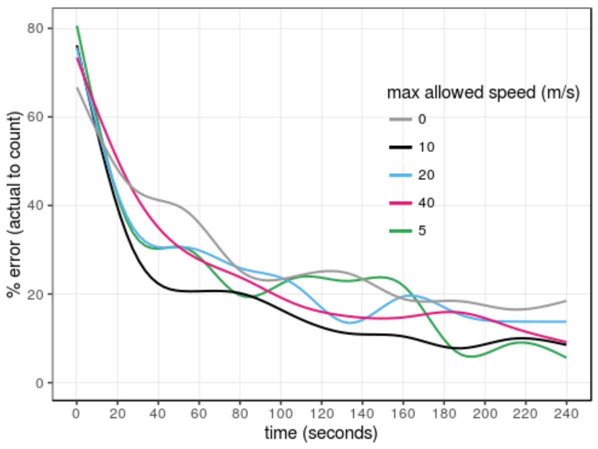


Figure 5.13: Mean counting error in randomly moving nodes

Figure 5.13 shows the average error in count in nodes inside the allowed counting grid. The algorithm performs well even under high mobility. Note that the error rate includes the islanding problem caused by unreachable nodes (as seen in 0 m/s line). We also observe that mobility in fact helps the counting process to bridge the possible island counts hence reducing error as compared to the stationary case.

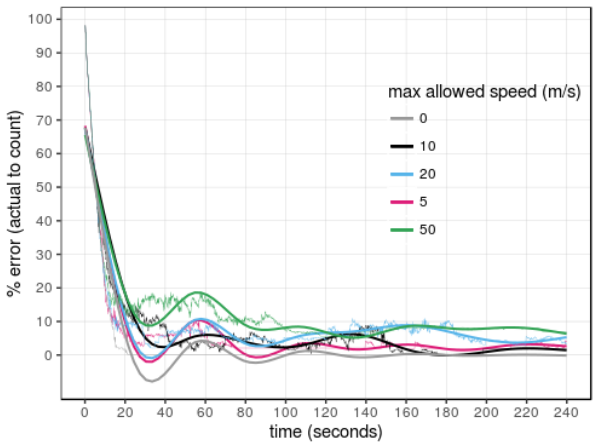


Figure 5.14: Mean counting error in nodes on a highway

Next, we setup a highway scenario. Nodes are placed on a stretch of $50m \times 10km$ section of a linear highway. Nodes are assigned speeds only along one axis (but in both directions). The chosen counting grid is the stretch between 2.5km and 7.5km (5km) section. Figure 5.14 shows the percentage error between actual number of nodes that have entered the section of highway and the estimate from counting process. It can be seen that since nodes are closer, islanding problem is minimized (negative error value is due to smoothening and should be considered zero). For moderate speeds convergence is achieved within 40 seconds. Furthermore, for all speeds the percentage error is below 5% indicating good performance of the algorithm.

5.4.2.3.3 Summary: Summarizing mobility versus counting, we conclude good algorithmic performance in mobile conditions. It was also seen that mobility helps connect isolated groups hence reducing individual errors. Highways due to their property of keeping nodes close, provide ideal conditions for counting.

5.4.2.4 Experiment 4: Multiple triggers and islanding

5.4.2.4.1 Setting: Until now, a single forced trigger was used to start counting. Here, the algorithm is made flexible and nodes start count themselves when they qualify. A simple extension to auto re-broadcast timer is used to achieve this. Nodes, periodically check for participation eligibility and when they qualify, they schedule a packet with initial counting parameters for broadcast. Hereafter, counting proceeds as usual.

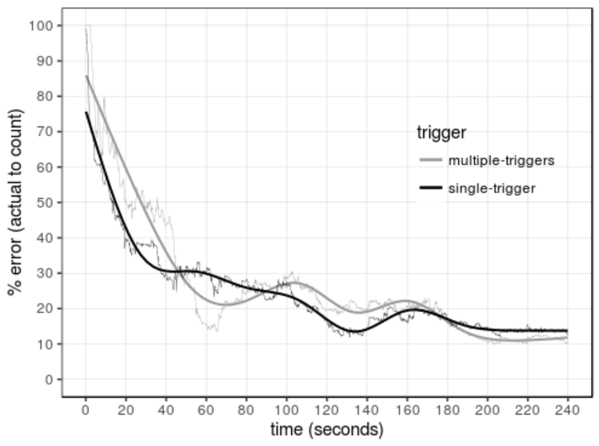


Figure 5.15: Comparison of count start triggers

5.4.2.4.2 Results: To compare single trigger and multiple trigger counting, we simply analyze count vs. actual cumulative nodes as in the previous experiment. Here, node speeds upto 20 m/s are allowed, with 100 nodes in $10km \times 10km$ space with counting allowed in the $5km \times 5km$ innermost space.

Figure 5.15 shows the comparison between single and multiple triggers. With multiple triggers allowed we expect nodes receiving triggers to start counting faster. We see that multiple triggers reduce errors quickly during initial stages of counting. Later, the performance settles with single trigger errors. The effect of having multiple triggers allows for autonomous decision making and fast counting. Single triggers are not practical in real scenarios (unless RSUs trigger counting).

To analyze islanding and its effect on counting, two islands are forced by grouping 15 nodes in each at the beginning of simulation as shown in Figure 5.16. Speeds upto 40 m/s are allowed in random directions for all nodes. We then measure the progress of counting by computing the median count of counts at all nodes.

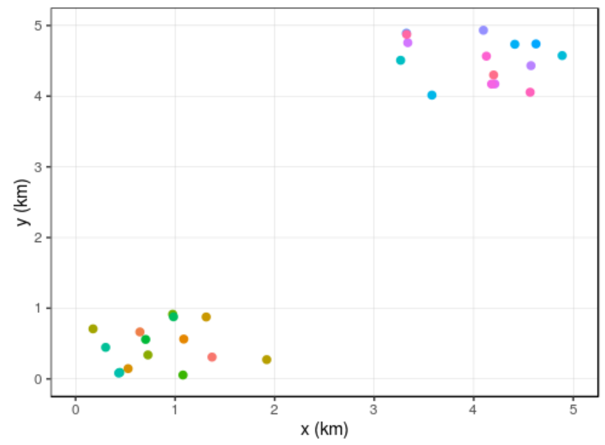


Figure 5.16: Islands of nodes at start

The counting progress is as shown in Figure 5.17. It is easy to observe how quickly median counts reach 15, the number of nodes in each island. Counting converges quickly inside the isolated islands (local convergence). And then as the nodes move, bridges are formed and the islands are merged. The “steps” seen in plots (also in previous experiments) is due to islanding of nodes.

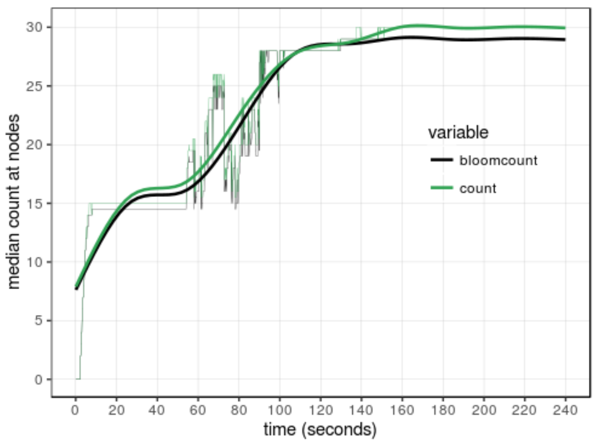


Figure 5.17: Counting with node islands

It can be seen from Figure 5.17 that, as bridges are formed, counts merge and the global count slowly rises. With time, the count reaches its final value, 30, which is the number of nodes in the grid. The two variables “count” and “bloomcount” correspond to list and bloom filter estimates. The error between the two is due to the probabilistic estimation of number of keys in a bloom filter as described earlier.

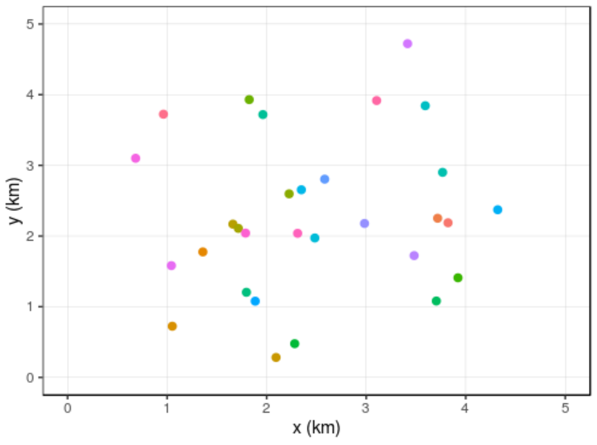


Figure 5.18: Nodes after 240s of counting

Figure 5.18 shows the state of node positions after 240 seconds. Clearly, the nodes have merged and there are no grouped islands hence confirming the expectation.

5.4.2.4.3 Summary: The effects of multiple counting triggers was observed to result in faster counting and was also discussed to be the feasible implementation method in real scenarios. The problem of islanding and its effect on counting progress, which is to form isolated counts, was studied by forced islands which merge with time.

5.5 Summary

In this chapter, data cycling and transience in wireless networks was analyzed and discussed. Next, vehicular networks and WAVE standards were introduced for studying mobile ad-hoc networks in detail. We then introduced an application of data cycling in mobile ad-hoc networks (vehicular) for counting vehicles in confined grids. The algorithm was implemented and simulations were done to study the various aspects of accumulative counting.

Although the application discussed here was specific to ad-hoc networked nodes, the concept of using data cycling is a general one. The counting application demonstrated the temporal context storage property of data cycling. Context or "state" storage is a broader problem and finds application in problems that can be solved using state machines. For example, the above accumulation problem can be generalized to an enclosing grid of grids where the local state maintained by data cycling can be used to paint a broader picture over many adjacent and enclosing grids.

Some aspects of WAVE, such as multi-channel operation and switching, emergency broadcasts were not tested as they were out of the scope of this study and they focus on the WAVE standard instead. However, since MAC operates a contention resolution queue for different access classes ($AC0$–3) in WAVE, higher priority messages will always succeed first. Hence, dramatic effects on counting is not expected, although there may be slight delays in convergence time.

Chapter 6

Conclusion and Discussion

6.1 Summary of Work

This study introduced the idea of setting up loops using network nodes which can be used to temporarily store useful data by continuously forwarding data until its usage (transient data). We started with a simple introduction to the idea using loops formed by garden hoses and then discussed the delays from the perspective of physics of communication. Next, an experiment was presented in which a folded light beam trapped between two parallel mirrors was used to "emulatate" temporary storage. We further drew parallels in optical networks where optical buffers are used for temporary storage.

After a brief background of packet switching, delays in today's computer networks were analyzed. Queuing and other data transmission delays were discussed as these form the basis of *how long* transient data exists on a given network link. Further we discussed the important idea of bandwidth delay product and LFNs which provides a quantitative picture of "in-flight" data on large and long networks such as satellite links.

Using this idea of transient storage, wired network loops were setup to continuously cycle data. We experimented some aspects of cycling data on wired networks and discussed the impacts of the same on network traffic. An application idea for smart devices and its advantages were briefly discussed.

Next, loops and data cycling on wireless networks and its differences from wired networks were discussed. An application of data cycling was then developed in wireless ad-hoc networks using real world WAVE protocol (vehicular networks). In depth analysis of the algorithm and how repetitive data forwarding maintained state (context) for new arriving nodes as transient data was studied. Detailed simulations demonstrated the working of this idea. The influence of various parameters on the performance of accumulative counting was simulated and discussed.

6.1.1 Contributions

The main contributions of this study are as below:

- Introduced the idea of transient data storage and data cycles in various networks.
- Studied data cycling qualitatively and quantitatively.
- Studied the impact of data cycling on network traffic.
- Introduced applications of controlled data cycling on networks for easy data accesses.
- Presented an application of transient data cycling for context maintenance and counting in vehicular networks. Detailed simulations and results were presented.

6.1.2 Uses and limitations of Data Cycling

The study revolves around transient data storage and data cycles in networks. Various aspects of data cycles were discussed and here they are compiled together.

"Storage" can be seen as holding data using some physical mechanism (or medium) until being used. We attempted to store data for short durations of time using networking delays and temporary buffers on nodes (as in wireless networks). Also discussed was optimizing data access using data cycles in applications where data is ephemerally useful and users of data are loss tolerant. Losses can be reduced by introducing redundancy. Data cycles and temporary storage are interesting in applications where central storage or control is not possible. An example of this, as discussed, is a fast changing network (like vehicular networks). Data cycles are also useful in avoiding central storage/access, but instead using temporary buffering and delays for storage.

In Chapter 4 we saw that cycling data has its disadvantages. First, it can cause lot of traffic on network links, especially if data cycling is continuous and unregulated. As a mitigating solution, backoffs and data purging (after invalidation) can help reduce unnecessary traffic. Second, since packet switched networks have a non-zero data loss probability, there is always a chance that data on cycles can be lost. Failure of network nodes, physical damages to network links, large amounts of extraneous noise can be possible causes of loss. However, it worth noting that system failures can also affect conventional storage (like hard drives). Power consumption is another very important aspect which was not studied in this work due to its introductory scope.

6.2 Scope for Future Work

Transient data and temporary storage in cycles can be studied further in detail and below are some points for future work:

1. Detailed mathematical analysis and modeling of data cycling on networks.
2. Power consumption due to data cycling and comparison with traditional storage and access.
3. Methods for introducing redundancy in transient storage to mitigate losses.
4. New applications of ephemeral data storage and usage.

Bibliography

[1] Hari Balakrishnan et al. *6.02 Handouts and Lecture notes.* URL: http://web.mit.edu/6.02/www/s2009/handouts/L17.pdf.

[2] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. "Sizing Router Buffers". In: *SIGCOMM Comput. Commun. Rev.* 34.4 (Aug. 2004), pp. 281–292. ISSN: 0146-4833. DOI: 10.1145/1030194.1015499. URL: http://doi.acm.org/10.1145/1030194.1015499.

[3] Burton H. Bloom. "Space/Time Trade-offs in Hash Coding with Allowable Errors". In: *Commun. ACM* 13.7 (July 1970), pp. 422–426. ISSN: 0001-0782. DOI: 10.1145/362686.362692. URL: http://doi.acm.org/10.1145/362686.362692.

[4] Pittsburgh Supercomputing Center. *Enabling High Performance Data Transfers.* URL: http://www.psc.edu/index.php/networking/641-tcp-tune.

[5] Kai Chen et al. "Understanding bandwidth-delay product in mobile ad hoc networks". In: *Computer Communications* 27.10 (2004), pp. 923–934.

[6] Inc. Cisco Systems. *Introduction to DWDM Technology.* URL: https://www.cisco.com/application/pdf/en/us/guest/products/ps2011/c2001/ccmigration_09186a00802342cf.pdf.

[7] NS-3 Consortium. *NIST Error Rate Model.* URL: https://www.nsnam.org/~pei/80211ofdm.pdf.

[8] NS-3 Consortium. *NS-3 Discrete Event Simulator.* URL: https://www.nsnam.org/overview/publications.

[9] NS-3 Consortium. *Two ray ground path loss model.* URL: https://www.nsnam.org/docs/release/3.25/models/html/propagation.html.

[10] Donald Watts Davies. *Oral history interview with Donald W. Davies. Charles Babbage Institute.* URL: `http://hdl.handle.net/11299/107241`.

[11] Ramakrishnan Durairajan et al. "InterTubes: A Study of the US Long-haul Fiber-optic Infrastructure". In: *SIGCOMM Comput. Commun. Rev.* 45.4 (Aug. 2015), pp. 565–578. ISSN: 0146-4833. DOI: `10.1145/2829988.2787499`. URL: `http://doi.acm.org/10.1145/2829988.2787499`.

[12] Stephan Eichler. "Performance evaluation of the IEEE 802.11 p WAVE communication standard". In: *2007 IEEE 66th Vehicular Technology Conference*. IEEE. 2007, pp. 2199–2203.

[13] John E. Bowers Emily F. Burmeister Daniel J. Blumenthal. *Optical Buffering for Next-Generation Routers*. URL: `http://optoelectronics.ece.ucsb.edu/sites/default/files/publications/burmeister06oec.pdf`.

[14] Li Fan et al. "Summary Cache: A Scalable Wide-area Web Cache Sharing Protocol". In: *IEEE/ACM Trans. Netw.* 8.3 (June 2000), pp. 281–293. ISSN: 1063-6692. DOI: `10.1109/90.851975`. URL: `http://dx.doi.org/10.1109/90.851975`.

[15] FCC. *Analysis of Satellite-Based Telecommunications and Broadband Services.* URL: `https://ecfsapi.fcc.gov/file/7520956711.pdf`.

[16] H. T. Friis. "A Note on a Simple Transmission Formula". In: *Proceedings of the IRE* 34.5 (May 1946), pp. 254–256. ISSN: 0096-8390. DOI: `10.1109/JRPROC.1946.234568`.

[17] G. Hernandez. *Fabry-Perot Interferometers (Cambridge Studies in Modern Optics)*. Cambridge University Press, 1988. ISBN: 052136812X.

[18] M.C. Hutley. "Single-Mode Fiber Optics: Principles and Applications". In: *Optica Acta: International Journal of Optics* 31.11 (1984), pp. 1209–1209. DOI: `10.1080/713821452`. URL: `http://dx.doi.org/10.1080/713821452`.

[19] Manish Jain, Ravi S Prasad, and Constantinos Dovrolis. "The TCP bandwidth-delay product revisited: network buffering, cross traffic, and socket buffer auto-sizing". In: (2003).

[20] Joseph M. Kahn. *Modulation and Detection Techniques for Optical Communication Systems.* URL: http://www-ee.stanford.edu/~jmk/pubs/mod.and.det.tech.COTA.6-06.pdf.

[21] Adam Kirsch and Michael Mitzenmacher. "Less Hashing, Same Performance: Building a Better Bloom Filter". In: *Random Struct. Algorithms* 33.2 (Sept. 2008), pp. 187–218. ISSN: 1042-9832. DOI: 10.1002/rsa.v33:2. URL: http://dx.doi.org/10.1002/rsa.v33:2.

[22] T. Komine and M. Nakagawa. "Fundamental analysis for visible-light communication system using LED lights". In: *IEEE Transactions on Consumer Electronics* 50.1 (Feb. 2004), pp. 100–107. ISSN: 0098-3063. DOI: 10.1109/TCE.2004.1277847.

[23] Hansuek Lee et al. "Ultra-low-loss optical delay line on a silicon chip". In: *Nature Communications* 3 (May 2012). Article, URL: http://dx.doi.org/10.1038/ncomms1876.

[24] L. Liao et al. "40 Gbit/s silicon optical modulator for highspeed applications". In: *Electronics Letters* 43.22 (Oct. 2007). ISSN: 0013-5194. DOI: 10.1049/el:20072253.

[25] Aiming Liu et al. "Dual-loop optical buffer (DLOB) based on a 3 times; 3 collinear fiber coupler". In: *IEEE Photonics Technology Letters* 16.9 (Sept. 2004), pp. 2129–2131. ISSN: 1041-1135. DOI: 10.1109/LPT.2004.833063.

[26] Brian Tierney Michael Smitasin. *Evaluating Network Buffer Size requirements for Very Large Data Transfers.* URL: http://www.es.net/assets/pubs_presos/NANOG64mnsmitasinbltierneybuffersize.pptx.pdf.

[27] LE Miller. "Validation of 802.11 a/UWB coexistence simulation". In: *national institute of standards and technology (NIST), WCTG white paper* (2003).

[28] Kenta Mori et al. "Experimental Study on Channel Congestion using IEEE 802.11 p Communication System". In: *IPSJ Technical Workshop on Mobile Computing and Ubiquitous Communications.* 2013.

[29] A. Paier et al. "Average Downstream Performance of Measured IEEE 802.11p Infrastructure-to-Vehicle Links". In: *2010 IEEE International Conference on Communications Workshops.* May 2010, pp. 1–5. DOI: 10.1109/ICCW.2010.5503934.

[30] L. G. Roberts. "The evolution of packet switching". In: *Proceedings of the IEEE* 66.11 (Nov. 1978), pp. 1307–1313. ISSN: 0018-9219. DOI: `10.1109/PROC.1978.11141`.

[31] S Joshua Swamidass and Pierre Baldi. "Mathematical correction for fingerprint similarity measures to improve chemical retrieval". In: *Journal of chemical information and modeling* 47.3 (2007), pp. 952–964.

[32] TeleGeography. *Submarine Cable Map.* URL: `http://www.submarinecablemap.com/`.

[33] Technology UK. *Packet-Switching.* URL: `http://www.technologyuk.net/telecommunications/communication-technologies/packet-switching.shtml`.

[34] R. A. Uzcategui, A. J. De Sucre, and G. Acosta-Marum. "Wave: A tutorial". In: *IEEE Communications Magazine* 47.5 (May 2009), pp. 126–133. ISSN: 0163-6804. DOI: `10.1109/MCOM.2009.4939288`.

[35] Jelena Vučić et al. "513 Mbit/s Visible Light Communications Link Based on DMT-Modulation of a White LED". In: *J. Lightwave Technol.* 28.24 (Dec. 2010), pp. 3512–3518. URL: `http://jlt.osa.org/abstract.cfm?URI=jlt-28-24-3512`.

[36] Y Wang et al. "Continuously-tunable dispersionless 44-ns all optical delay element using a two-pump PPLN, DCF, and a dispersion compensator". In: *Signal* 16.26 (2005), p. 36.

[37] Jim Warner. *Buffer Sizes.* URL: `https://people.ucsc.edu/~warner/buffer.html`.

[38] Jim Warner. *Evaluating Network Buffer Size requirements for Very Large Data Transfers.* URL: `https://people.ucsc.edu/~warner/Bufs/buffer-requirements`.

[39] G. P. Watson et al. "Spatial light modulator for maskless optical projection lithography". In: *Journal of Vacuum Science & Technology B* 24.6 (2006), pp. 2852–2856. DOI: `http://dx.doi.org/10.1116/1.2387156`. URL: `http://scitation.aip.org/content/avs/journal/jvstb/24/6/10.1116/1.2387156`.

[40] Wikipedia. *Murmur Hash.* URL: `https://en.wikipedia.org/wiki/MurmurHash#Algorithm`.

[41] Shun Yao, Biswanath Mukherjee, and Sudhir Dixit. “Advances in photonic packet switching: an overview”. In: *IEEE Communications Magazine* 38.2 (2000), pp. 84–94.

[42] Yong-Kee Yeo, YU JIANJUN, and Gee-Kung Chang. “A dynamically reconfigurable folded-path time delay buffer for optical packet switching”. In: *IEEE photonics technology letters* 16.11 (2004), pp. 2559–2561.

[43] Xiaoming Zhu and Joseph M Kahn. “Queueing models of optical delay lines in synchronous and asynchronous optical packet-switching networks”. In: *Optical Engineering* 42.6 (2003), pp. 1741–1748.

Appendix A

Glossary and Acronyms

Care has been taken in this thesis to minimize the use of jargon and acronyms, but this cannot always be achieved. This appendix defines jargon terms in a glossary, and contains a table of acronyms and their meaning.

A.1 Glossary

- **Data Cycle** – Data circulated or forwarded continuously on a loop formed by network nodes.
- **Transient Data** – Data that is "in-flight" on a network. In other words, data that has been sent but not received by the intended receiver.
- **Bandwidth Delay Product** – Easy. The product of network bandwidth (in bits per second) and the round trip time of the network segment.
- **Long Fat Network (LFN)** – A network linkthat has massive communication bandwidth (order of several terabits per second) and also has a vast geographical prominence, hence resulting in a large bandwidth delay product.

A.2 Acronyms

Table A.1: Acronyms

Acronym	Meaning
WAN	Wide Area Network
LFN	Long Fat Network
BW	Bandwidth
RTT	Round Trip Time
ARPANET	Advanced Research Projects Agency Network
PoP	Point of Presence
PSTN	Public Switched Telephone Network
ISDN	Integrated Services Digital Network
ISP	Internet Service Provider
IP	Internet Protocol
IoT	Internet of Things
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
DWDM	Dense Wavelength Division Multiplexing
OFDM	Orthogonal Frequency Division Multiplexing
WAVE	Wireless Access for Vehicular Environments
SCH	Service Channel
CCH	Control Channel
GPS	Global Positioning System
RSU	Road Side Unit
OBU	On Board Unit
MAC	Medium Access Control
VANET	Vehicular Adhoc Networks
MANET	Mobile Adhoc Networks
DSRC	Dedicated Short Range Communication
FCC	Federal Communications Commission
EDCA	Enhanced Distributed Channel Access
DCF	Distributed Control Function