Scalable Routing for Mechanical Backhaul Networks

A Thesis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Master of Technology

by

Zahir Koradia

to the

Department of Computer Science & Engineering

Indian Institute of Technology, Kanpur

June 2007

Certificate

This is to certify that the work contained in the thesis entitled "*Scalable Routing for Mechanical Backhaul Networks*" by *Zahir Koradia* has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

(Dr. Bhaskaran Raman)

Department of Computer Science & Engineering,

Indian Institute of Technology,

Kanpur-208016.

Abstract

Providing Internet connectivity to rural regions in developing countries is a challenging proposition. To effectively serve the needs of the rural poor, solutions need to be low-cost, in addition to being reliable and scalable. Mechanical backhaul networks are a low-cost solution to providing delay tolerant connectivity to such regions. In these networks, shared kiosk-based computers in villages connect opportunistically to wireless router nodes present on buses and other similar vehicles. These vehicles “mechanically” ferry data between kiosks and Internet gateways. With only as much as a minute of contact duration, the ferry can transfer several tens of MB of data over a wireless link.

In this work, we design, implement, and evaluate a protocol for *scalable* and *robust* routing in such a setting. We achieve scaling by the appropriate use of hierarchy in routing; and we achieve robustness through the use of a smart flooding approach. During our design, we also take into consideration constraints peculiar to our problem setting, which include use of low power, low speed machines to keep the costs low, availability of low speed (100 Kbps) Internet connections at various locations, and availability of cheap non-volatile storage.

Using a combination of simulations and a prototype implementation, we show that our solution can scale to large-scale, country-wide deployments. Further, we show that for the problem setting under consideration, with a network size of 100,000 nodes, a kiosk-based computer can be allowed close to $260MB$ of upload per month. This upload capacity is more than sufficient for applications like e-mail, offline web browsing, e-governance services, transfer of agricultural knowledge to rural regions, etc. We identify and analyze various aspects of the problem setting that can be bottlenecks for the system performance. We show that low power machines form the primary bottleneck of performance in our solution.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Provision of Internet connectivity in rural regions of developing countries in a cost-effective manner is a pragmatic solution to bridging the digital divide. By opening up access to knowledge and making heard the voice of the masses, bidirectional information access can open up new possibilities [6]. However, providing robust, low-cost connectivity is a challenging proposition [9] [5]. Several competing technologies, such as long-range WiFi, WiMax, and 3G have been proposed for this purpose.

1.1 Motivation and Background

Recently, mechanical backhaul networks have been shown to be a low-cost alternative to provide delay-tolerant connectivity to rural and remote regions [16] [15] [10], [19]. Mechanical backhaul networks are a part of the general class of *Delay Tolerant Networks (DTN).* To understand what delay tolerant networks are let us start with an example scenario. Take a computer that is not connected to the Internet. Open an e-mail client and try to send an e-mail to some address. You will get an error stating that it could not connect to the mail server. This is quite expected. The point to be emphasized is that the mail client tries to connect to the mail server for a few times and eventually times out. Now assume that this computer is sitting in some remote village with no direct Internet connection. However, a bus traveling between the village and the city has an access point sitting on it. This means we can dump the mail on to the machine on the bus and the bus can carry the data for us to the city and drop it on to a node connected to the Internet. This node can then send the mail for us. Similarly, if there is an e-mail for us then the node on the Internet collects it and sends it to us in the village via the bus.

Notice that there is no direct connectivity from the machine sitting in the village to the Internet. Yet it is possible for us to be able to communicate to the Internet. In the same manner we could have a network of nodes which are connected via buses and the Internet. The current Internet with its protocols cannot handle such a situation primarily because of one basic assumption: the delay between source and destination for any kind of communication is of the order of milli-seconds or at most a few seconds. Along with this TCP, the most dominant transport layer protocol on the Internet, requires end to end connectivity. This is where the concept of delay tolerant networking (DTN) comes in.

Delay tolerant networks are also synonymously called disruption tolerant networks. The scenario presented above is not the primary motivation behind DTN. The focus of the DTN Research Group [1], while working on DTN has been deep space communication with large delays. Certain mobile wireless networks and sensor network applications can also be modeled as DTN. For example ZebraNet [14] involves transfer of data from the collars fixed on zebras to base stations, where both former and latter are mobile. The links between two zebras or a zebra and a base station becomes available *opportunistically*. Such situation can be modeled as a DTN where the zebra collars and the base stations are DTN nodes and routing of data must be done from the zebra collars to the base stations as and when links become available. To define DTN more formally, Delay Tolerant Networks are the networks that are robust to disruption and large delays in connectivity between two communicating nodes. DTN can further be categorized into occasionally connected and occasionally disconnected networks. A wireless mobile network with intermittent disconnectivities can be a representative of occasionally disconnected networks while the bus-kiosk networks discussed above can be considered occasionally connected networks.

The concept of delay tolerant networking brings to the fore new challenges. Firstly, permanent storage at source and intermediate hops is required to be able to save data for long times across disconnections. Secondly, routing becomes an issue with the weighted multigraph abstraction of the Internet getting converted to time varying weighted multigraph in DTN scenarios. It is important to note that poor routing decisions can lead to huge amounts of delays as the networks can be disconnected. Exchange of information for the purpose of routing becomes difficult since there is no fast way to exchange information and have globally consistent routing tables. Deciding when to retransmit a piece of data is another tough question to answer. We also want the nodes connected through DTN to be able to communicate to nodes on the Internet. For this we need proxy nodes acting as rendezvous points between the DTN nodes and the Internet. All these issues make DTN challenging.

1.2 Mechanical Backhaul Networks

In spite of the challenges faced by DTN mechanical backhaul networks have been shown to have the potential of providing low cost delay tolerant connectivity to rural regions. In such networks, kiosks are set up near bus stops in villages. These kiosks are accessed by citizens living in and around the village for services like email, access to agricultural information, e-government services and so on. Buses (and other similar vehicles) carry mobile routers, which transfer data to and from the kiosks over IEEE 802.11 [2] links when the buses stop near (or even merely go past) a kiosk. These mobile routers are typically low power, low cost single board computers, with high volume of secondary storage. It has already been observed that even with only a minute or so of connectivity, several tens of megabytes of data can be transferred between two nodes [15].

The mobile router ferries data to and from *Internet gateways*, which are wireless routers that have an Internet connection. Gateways are typically connected to the Internet through low-to-medium bandwidth technologies like ADSL [16] with a connection speed of the order of 100 Kbps. This translates to approximately 1GB per day.

A schematic of a mechanical back-haul network is shown in Fig. 1.1. As shown in the figure, there can be many buses connecting a given kiosk to one or more gateways and, symmetrically, there can be many kiosks connected by one bus to one or more gateways.

Many applications can use such a delay-tolerant network. Examples include email, offline web browsing, e-governance, and transfer of knowledge from agricultural experts to farmers. We believe that such applications can help to improve the quality of life in rural regions of the world [5].

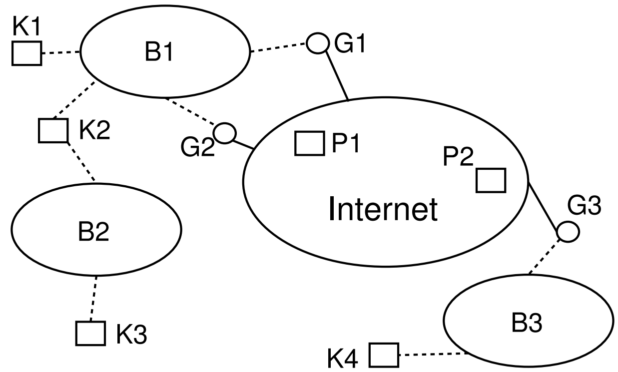


Figure 1.1: A Bus-Kiosk Network. K: Kiosks, B: Buses, G: Gateways. Ovals around buses indicate bus routes. Dashed lines indicate occasionally available WiFi links. Solid lines indicate wired connections.

1.3 Problem Statement and Thesis Contributions

Given the description of mechanical backhaul networks above, our goal in this thesis is to enable (a) routing of data between a kiosk and a node on the Internet, as well as (b) routing data when both source and destination are kiosks.

We address two critical challenges in the design of a routing protocol for such networks. (1) *Scalability* of operation. This is particularly important; for instance, a country-wide deployment of tens of thousands of kiosks, all over India, is being planned over the next few years [3]. (2) *Robustness* in data delivery. That is, the network must deliver each "bundle" from source to destination with high probability.

Our essential approach to address scalability is the use of hierarchy: we divide the network into several *regions*. We then address intra-region routing and inter-region routing separately. To achieve robustness, we use a smart flooding approach, within a region. This ensures that a data bundle reaches from a kiosk to a gateway (or vice versa) with very high probability. We argue that the use of flooding is a good design choice, and does *not* affect system scalability. Further, we also identify and analyze various aspects of the problem setting which have the potential of becoming bottlenecks for the performance of the our solution. The contributions of this thesis are:

1. Design, implementation, and analysis of a scalable routing protocol for mechanical backhaul networks.

2. Use of *smart flooding* for routing to achieve robustness along with satisfactory system performance.

3. Consideration of characteristics peculiar to rural setting into the system design.

4. Analysis of potential bottlenecks in the given problem setting of rural deployment.

1.4 Thesis Overview

The rest of the thesis is organized as follows. In the next chapter we present our architecture for scalable and robust routing. Further, in the same chapter, we elaborates on a few important design and implementation issues. We undertake an evaluation of our design in Chapter 3. Chapter 4 describes prior work related to ours, and Chapter 5 concludes the report.

Chapter 2

Architecture for Scalable Routing

We have designed the our routing architecture to be scalable so as to be able to deploy it as a country wide network. The architecture makes appropriate use of hierarchy to achieve scalability. It also achieves high robustness by making use of replication and adding redundancy. Further, the the architecture takes into account the peculiar characteristics of rural deployments like low cost requirement and availability of *lowspeed* Internet connections. As a result we are able to utilize abundant resources for adding robustness to the system while avoiding usage of scarce resources. Now we describe the scalable routing architecture in greater detail.

2.1 Architecture Overview

A schematic diagram of the scalable routing architecture is shown in Fig. 2.1. We model our architecture as a Delay Tolerant Network(DTN) [7] because we face similar connectivity constraints as a DTN, that is no end-to-end connectivity and intermittent availability of links. To achieve scalability we partition the set of kiosks into *regions*. A region consists of (a) a number of kiosk-based computers located at villages that are accessed for various services, (b) a number of gateways, where a gateway refers to a router that has at least one interface on the Internet, and (c) a number of buses carrying wireless mobile routers that ferry between the kiosks and the gateways. The actual method used to create region boundaries is left to the network administrator's discretion and can be chosen to optimize the situation at hand. We only require that all the nodes in a region are connected; that is, if we considered buses as edges and other nodes as vertices of a multigraph, then the multigraph must be connected. One general guideline that can be used while defining regions is the geographical proximity of the nodes. One way to mark regions would be along geographical boundaries of states/districts, which would also be administrative boundaries.

A region can have multiple gateways. The gateways of a region act as points-of-contact between the region and other regions. We designate a *proxy* for each region. This proxy is responsible for interfacing between nodes that are aware of Delay Tolerant Network (DTN) architecture and nodes that are unaware of it. For example, a kiosk trying to send data to a host on the Internet needs some mechanism to translate its information present in bundles (the basic data unit in DTN architecture) to IP-format packets. The proxy is responsible for such a translation.

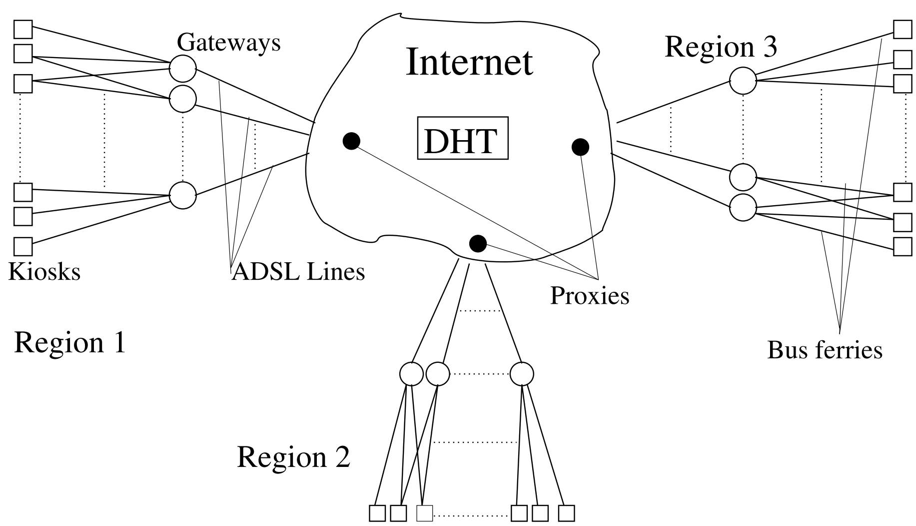


Figure 2.1: Scalable Routing Architecture

We use global unique addresses as described in [18] because that allows us to give a location-independent identity to each user, process, and node. Each entity, which could be a user, a process or a machine, is given a globally unique address. This address is human readable and acts as an identification for an entity. The format of the address is given as *tca://userid.machineid.regionid.kiosknet.net/application*. Although we need the addresses to be unique we do not incorporate any mechanism for ensuring unique addresses. However, we believe this can be ensured administratively. For example, giving a unique *regionid* to each region ensures that all addresses from one region can be differentiated from another. Within a region we could use the IP addresses of the machines as their *machineids*. Finally, it only remains to ensure that all the users using the same machine have different *userids*, which can be ensured by the kiosk/gateway operators.

A global routing table implemented as a distributed hash table (DHT) stores entries for each node on the network indicating to which region a node currently belongs. Hence, it provides a flat namespace-to-location mapping. The use of a DHT provides (a) scalability: large number of requests can be handled bye the global routing table and (b) availability: downtime for the routing table is reduced due to distributed nature of the DHT. We give up on atomicity of update operations on the routing table. However, later in Section 2.5.3 we describe how we deal with this issue. An entry for a node is made in the DHT through a process called registration described in Section 2.5.1.

With this architecture in hand we now describe our solution to the problem of routing in four different scenarios: (1) When source and destination are in the same region, (2) When source and destination are in different regions, (3) When the source is in one region and the destination is on the Internet, and (4) When the source is on the Internet and destination is in some region. We also discuss some other issues that arise out of the architecture design.

2.2 Intra-Region Routing

We first consider intra region routing, i.e. routing between two kiosks of the same region. The challenge here is in find the *best* path between a source and a destination. For this purpose the source node must know (a) whether the destination is in the same region and (b) on which of its link should it send the data in order to take the *best* path to the destination. These two depend on the definition of *best* path. We define *best* path from a source to a destination as the one that delivers the bundle fast from the source to the destination. Note that by this definition the *best* may vary based on when does the source want to send data to the destination. Before we describe out solution a few characteristics of the problem setting are in order.

Firstly, note that we have large storage capacities, on the order of tens of GB, both at buses and at kiosks. Second, communication amongst buses, kiosks, and gateways is high-capacity. To get a sense of this, if we assume a nominal application throughput of 10Mbps for a 802.11g link and 1 minute as stoppage/transit time for a bus, we can transfer 600Mb or 75MB of application layer data in that span of time. Finally, communication on each wireless link is inexpensive because the communication is between two nodes on unlicensed spectrum, and does not involve a service provider, who may charge for the communication. Indeed, the only expense is that of power, which is easily available on buses using the existing battery, and at kiosks, using solar cells or diesel generators. Given these considerations, we choose to expend resources freely to obtain robustness in communication by transferring data *within* a region through $flooding$.

With flooding, when a kiosk or a bus has a bundle to send it simply floods the bundle on all available links as and when they become available. When a kiosk or a bus receives a bundle that is not destined to it, it floods the bundle on all the links as and when they become available. This means that both kiosks and node present on the buses act as routers. Gateways also flood bundles, but slightly differently: they flood a bundle only on the links which go to a bus or a kiosk and never on a link connecting it to the Internet. This minimizes the use of the Internet connection at the gateways. Moreover, duplicate bundles are dropped and not flooded a second time by any node. This process guarantees that all the nodes in the region will receive every bundle, as long as it has not expired due its time-to-live. Thus if the bundle destination was a node in the region then it will receive the bundle.

This also acts as a robust mechanism to get the bundle delivered to gateways from where further routing can take place. Note that a bundle is flooded whether or not the bundle destination belongs to the same region. This means that the source of the bundle need not know the region to which the destination belongs. Hence, no intelligent routing or book keeping is required in the kiosks and buses, greatly reducing implementation complexity. Also note that because regions are mutually exclusive, flooding within a region does not over flow into another region. Finally, since the bundle is sent over all the paths it will also be sent over the *best* path. Thus we have achieved our aim of sending the bundle from the source to the destination over the best path without having to compute it.

Note that in delay-tolerant networks in general, and in mechanical backhaul networks in particular, links are not always available. Therefore, to send a bundle on all the links associated with it, a router may need to wait for links to come up over time. Moreover, a mobile router may encounter new links that were created $after$ a bundle was received at the router. To send a bundle to all its links, a router stores an incoming bundle in its non-volatile storage until it expires. When a link becomes available (whether or not it existed at the time of bundle reception) the router sends a copy of the bundle on that link.

Smart flooding approaches

The flooding algorithm described above makes suboptimal use of storage space and network bandwidth. Firstly, when the source and the destination of a bundle are in the same region then once the bundle reaches the destination flooding can be terminated. Similarly, if the bundle is destined for another region then, once the gateway has sent the bundle to its destination region it need not be flooded any further in the region where the bundle originated. Secondly, if the bundle has been delivered to its destination (if in the same region) or destination region (if destination is in another region) the nodes can safely delete the bundle. Thirdly, although on reception of a duplicate bundle it is promptly discarded, network bandwidth is wasted in such an unnecessary transfer. Finally, duplicate bundle detection itself can take **O(m*n)** time, where m and n indicate number of bundles in the two nodes. Such a high processing requirement has the potential to become a bottleneck since we use low power routing modules to minimize costs of rural deployments. To deal with these issues we propose two orthogonal approaches to flood bundles in a *smart* manner.

When a bundle has reached its destination (in case where source and destination are in the same region) or when a gateway has delivered a bundle to its destination region, the destination node/gateway generates a *Death Certificate (DC)* for that bundle [8]. A death certificate is a special kind of control bundle. It contains the identity of the bundle for which it was generated. A death certificate indicates that the bundle to which it corresponds has reached its destination (or destination region) and that it need not be flooded any further in the region. This death certificate is then flooded in the region. When a node receives a death certificate if the bundle corresponding to the death certificate exits in its store it deletes the bundle. It then stores the death certificate irrespective of whether it had the bundle corresponding to the death certificate in its store. If a node receives a bundle for which it already has a death certificate in its store it discards the bundle. Death certificates are also flooded like any other data bundle. Since the size of a death certificate is expected to be much smaller than a data bundle it will save storage space on router nodes. Further, after receiving a death certificate for a bundle, a node will flood only the death certificate and not the bundle. This can also help optimize the usage of network bandwidth. However, all these arguments depend on the size of the data bundle for which the death certificate is being generated. If the data bundle itself is smaller than the death certificate its advantages are rendered useless.

To address the issue of duplicate bundles causing under-utilization of network bandwidth, we propose a phase of *Metadata Exchange (ME)* before actual data transfer when a link between two nodes becomes available. During this ME phase both the nodes exchange information regarding which bundles are not present with each node that the other can send. Once each node knows which bundles are not present with the other node, each node sends only those bundles to the other node. However, the ME process can still take **O(m*n)** time. To overcome the problem we store the bundles in the nodes as a sorted list. Each bundle can be uniquely identifies by its source address and creation timestamp. We create a unique key for each bundle by computing SHA1 hash for each bundle providing its source address and creation time stamp as the *digest*. The bundles stored in the list are ordered using this key. When a link between two nodes becomes available one of the two nodes, say K_1, sends the SHA1 hash of all the bundles in its list to the other node, say B_1. Note that this list of SHA1 hash is sorted. Now B_1 has information about all the bundles present in K_1's list. It uses this information along with its own list. Using this information B_1 can identify bundles it needs as well as the bundles that it can send to K1 in **O(m+n)** time. B1 then provides this information to K1. Note that with ME duplicate bundles are never transmitted. This saves processing time and network bandwidth.

It is important to point out that ME and DC attack different aspects of naive flooding protocol described above. DC tries to flush out data bundles that have been delivered to their destinations (or destination regions); and ME prevents transfer of duplicate bundles between nodes, eliminating the need for duplicate detection and saving network bandwidth. Hence, it is possible to have both these mechanisms working in tandem to achieve *smart flooding*. In Chapter 3 we study the effect of use of ME and DC in flooding and show that they are both effective techniques.

2.3 Inter-Region Routing

The primary challenge for inter-region routing is delivery of the bundle from the gateway of the source region to one of the gateways of the destination region. We consider four different approaches shown in Fig. 2.2 to Fig. 2.5. We discuss each of these below.

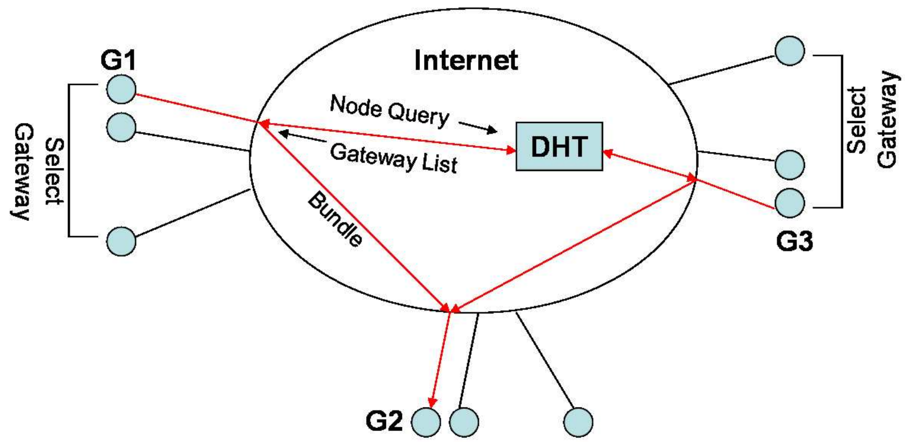


Figure 2.2: Gateway to Gateway (G2G)

2.3.1 Gateway to Gateway (G2G)

In this approach, when a gateway receives a bundle from a node of its region it first checks if the bundle has already been forwarded. This is necessary to ensure that a bundle is sent over the bottleneck link between the gateway and the Internet only once. This check is done by keeping a record of all the bundles that have been sent by all other gateways of that region. If the bundle has not been forwarded then it takes the responsibility of sending the bundle. It informs other gateways in its region of this decision. The other gateways keep a record of this information so that they do not forward the same bundle on receiving it. If two or more gateways decide to forward the same bundle at the same time then we could use a mechanism so that all gateways agree on the same gateway to send the bundle. The mechanism of achieving this *consensus* is discussed in Section 2.5.3. Once a gateway has been agreed upon then this *agreed upon gateway* obtains the list of possible destination gateways from the DHT. It also communicates with all other gateways, across all the regions, wanting to send bundles to the destination kiosk. It forms a consensus with them based on bus schedules in destination region and decides on which of the gateways in the destination region will it send the bundle. This decision making is the same as that in [11], which is described in Section 4. The primary difference is that in out case the decision making is done in a distributed manner across all the gateways wanting to send a bundle to that region. The gateway in the destination region that received the bundle can then send the bundle on the next bus to the destination using flooding, as described earlier.

The G2G solution is depicted in Fig. 2.2. When G1 receives a bundle, it checks its local records to confirm that the bundle has not be forwarded as yet. Then it communicates with the other gateways of its own region to inform them that it will send the bundle to its destination region. After necessary consensus amongst the gateways wanting send data to the same destination as G1 it sends the bundle to the destination gateway (say G2), which then floods the bundle in its region.

This solution has good performance because we do not have any bottlenecks. It is also reliable because we have multiple source gateways and multiple destination gateways and failure of any one of them does not affect the delivery of the bundle. However, the solution has very high complexity. Even making a centralized scheduling decision to choose the best gateway has turned out to be difficult [11] and in our case the decision making is distributed, which makes it even more difficult. Further, communicating with all the gateways of all the other regions would mean extensive communication overhead, which the bottleneck links of the gateways cannot handle.

Of course, one easy way out is to get a list of destination gateways from the DHT, randomly choose one of them and send the bundle to that gateway. That gateway can then flood the bundle in the destination region similar to the flooding in the source region. This is clearly not the best solution, because the selected gateway may be very far from the intended kiosk and random selection may lead to temporary overloading of gateways. In Section 3 we study the imbalance observed by gateways when destination gateways are selected at random. This solution can be used as a benchmark for comparison with alternative solutions.

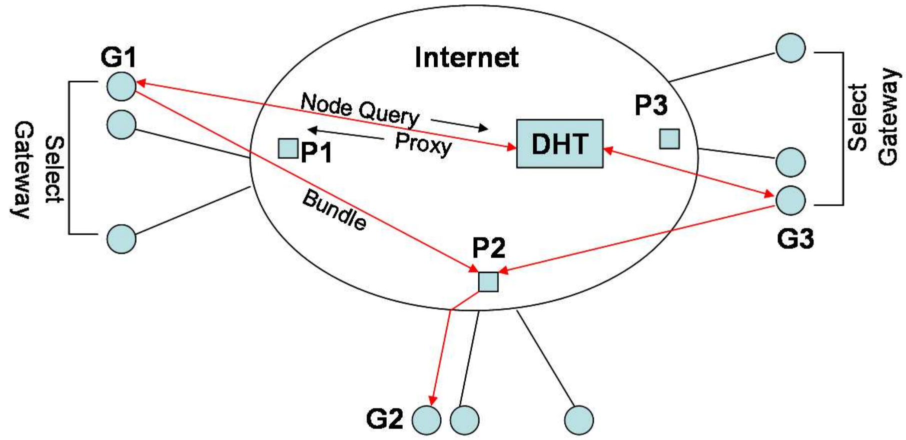


Figure 2.3: Gateway to Proxy (G2P)

2.3.2 Gateway to Proxy (G2P)

Here, when a gateway receives a bundle from a node of its region it first checks if the bundle has been forwarded. If it has been forwarded it drops the bundle else it obtains the address of the proxy of the destination region and sends the bundle to it. The proxy schedules the bundle to one of the gateways using the algorithm in [11]. Fig. 2.3 describes this solution. On receiving a bundle G1 first confirms that the bundle has not been already forwarded. Next a consensus is achieved amongst the gateways of the region that G1 will forward the bundle. G1 then gets the proxy associated with the destination region, which in this case is P2. Then P2 uses algorithm described in [11] and chooses G2 as the gateway to which the bundle should be sent.

Unfortunately, we may have a performance bottleneck in the form of the proxy here. Using high performance machines as proxy alleviates the issue to some extent but the bottleneck will still remain. The question of reliability also arises with one node responsible for delivering data to the whole region. However, there is not much complexity in the solution as no distributed decision making is required for choosing the gateway to which the bundle should be sent.

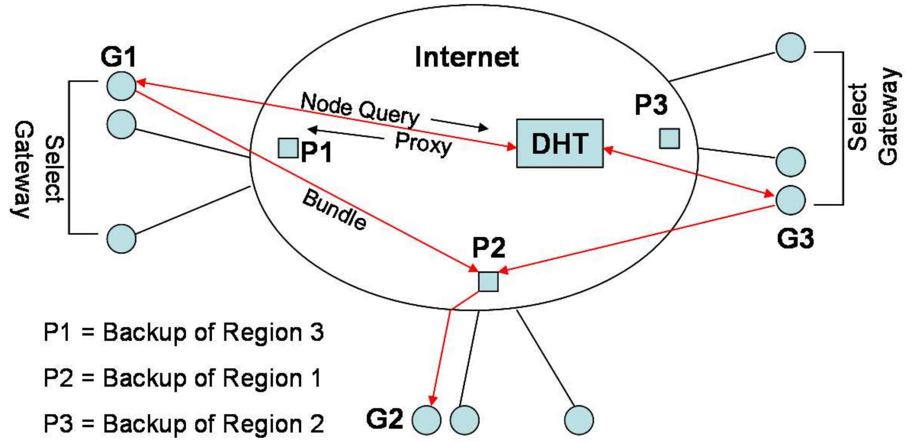


Figure 2.4: Gateway to Proxy with Backup (G2PB)

2.3.3 Gateway to Proxy with backup (G2PB)

This solution is the same as the G2P approach except that a primary proxy for one region also acts as a backup proxy for another region. Figure 2.4 gives a better view of this approach. In this figure P2 is the backup proxy for Region 1, P3 for Region 2 and P1 for Region 3. This means P2 remains aware of the state of P1, P3 remains aware of state of P2, and P1 remains aware of the state of P3. If P1 goes down then all the bundles destined to Region 1 will now be sent to P2. P2 will need to be aware of bus schedules of Region 1 and it will be the responsibility of P2 to deliver the bundles to Region 1 till P1 comes up again.

This approach provides sufficient robustness against proxy failures. Of course, with this solution, the performance bottleneck still exists; in fact, it worsens because now a proxy needs to be updated about its own state as well as for the region it is backing up. Moreover, there is additional complexity involved in this solution since a proxy would need to know the schedules of two regions and have routing information about two regions. When a proxy goes down some level of complexity is involved in making sure that the backup proxy takes up the responsibility. Nevertheless, in spite of the complexities involved the solution is much simpler than the G2G approach.

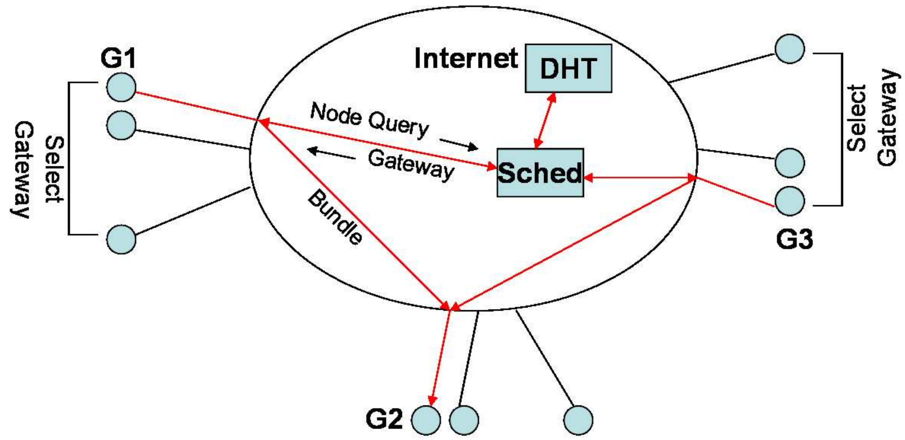


Figure 2.5: Gateway to Gateway with Centralized Scheduler (G2GC)

2.3.4 Gateway to Gateway with Centralized Scheduling (G2GC)

In this approach, when a gateway gets a bundle from its region, it checks as before if the bundle has already been sent. If not, then it informs other gateways that it will send the bundle. It then sends a request to a *centralized scheduler* stating the destination address. The scheduler gets such requests from all the regions for bundles destined to all the regions. It can know the region of the destination by looking it up in the DHT. It then sorts requests based on the region to which the bundle is destined. Then for each region, it schedules the bundles to one of the destination gateways using the algorithms described in [11]. Fig. 2.5 depicts an example scenario that uses the G2GC approach. G1, after reaching a consensus with other gateways of its region, requests the scheduler to give the gateway to which it should send the bundle. The scheduler replies to G1 informing it to send the bundle to G2. G1 sends the bundle to G2, which then floods the bundle in the destination region.

This approach does not avoid the bottlenecks and reliability issues related to proxies; rather it only transfers them to the scheduler. It is, however, possible to make one logical node robust by having a distributed deployment for the node. Nevertheless, performance issues still remain. From an implementation point of view it is rather complex to deploy a distributed node with such high processing requirements, needing to schedule every bundle destined outside the source region and process the request of each bundle. We can reduce the burden of the scheduler to a certain extent by making the gateways query the DHT to find out the destination region of the bundle it has to send and make it send a request to the scheduler only when the destination region is different from its own. However, this reduces performance requirements only marginally.

Solutions	Performance	Reliability	Complexity
G2G	high	high	high
G2P	low	low	low
G2PB	low	high	medium
G2GC	low	high	medium

Table 2.1: Comparison of possible solutions to inter region routing

Table 2.1 compares these solutions based on performance, reliability and complexity of implementation. G2GC and G2PB seem to be the winners amongst the four but the want of infrastructure for a distributed implementation of the scheduler in G2GC makes G2PB the most feasible solution.

2.4 Routing Between a Region and the Internet

When the destination of the bundle is a node on the Internet that is not aware of DTN then the bundle's destination is set to the proxy and it is the proxy's responsibility to do further processing. The application is expected to know that the other end of the communication is DTN-unaware. This expectation is not very demanding since it is the application that specifies the destination address and will not have a DTN address for the destination if it is DTN unaware.

If the destination node on the Internet is DTN aware then the bundle's destination must have an entry in the DHT. Since the destination node is on the Internet it can be represented as a region with just one node and the destination node as its gateway. With this approach any of the inter-region routing solutions can be used to deliver the bundle.

In the situation where source node is on the Internet, if the source node is DTN aware then it looks up the set of destination gateways in the global routing table and sends the bundle to one of the gateways. If the source node is not DTN aware then it cannot initiate communication to a kiosk since it will not be aware of the existence of the kiosk. Note that communication between a kiosk and a DTN unaware node can take place only through the proxy. Further, the communication must be initiated by the kiosk, in which case proxy communicates to the DTN unaware node on behalf of the kiosk.

2.5 Other Design Issues

2.5.1 Registration

We now look at the process of registration of a node in a region. For a node to receive bundles it is necessary that an entry in the global routing table (DHT) exists. To create this entry the node sends a *Register (REG)* bundle. This bundle is also flooded like any other bundle in the region. Notice that this means that bundles which are a part of the *control plane* are also being flooded. Although this is counter intuitive, the flooding process helps the *REG* bundles to reach all the gateways of the region. Which gateway informs the DHT about a registering node depends on the solution being used. If G2G or G2GC is used then all the gateways will send the information to the DHT so that the DHT will have the list of all gateways associated with that node. If G2P or G2PB is used the registration bundle is sent to the proxy of the region, which informs the DHT about the node.

2.5.2 Mobility management

If a node has moved from one region to another then the entry in the DHT corresponding to its previous region needs to be modified. For this purpose the node again sends a *REG* bundle initiating the registration process. Note that there is a possibility that bundles destined to the node that have not yet been delivered to it may be present in the old region. These bundles need to be sent to the new region. To do so, during the registration process, when an entry corresponding to old region for a node is already present in the DHT, the gateway responsible for making a new entry in the DHT selects one gateway at random from the list of gateways of the old region present in the DHT, and sends a *Change Of Address (COA)* bundle to the gateway (in case of G2P or G2PB the gateway of the new region sends the *COA* bundle to the proxy of the old region, which then chooses a gateway at random from the gateways of the old region). The gateway that receives the *COA* bundle sends all the bundles that are destined to the registering node and that are present in its store to the new region. Note also that before sending the *COA* the DHT entry is modified to indicate the new region of the registering node. This information can then be used by the gateway of the old region to forward the bundles destined to the mobile node. However, the gateway of the old region may not have all the bundles destined to the registering node as yet. To make sure that all the bundles destined to the registering node and delivered to the old region get forwarded to the new region, the gateway continues to forward all future bundles for that node that it receives from within its region. Since the gateway that sent a *COA* bundle to this gateway must have already updated the DHT entry no new bundles destined for the registering node are expected to arrive from another region.

Note that the solution is not the best one because it imposes a heavy load on the gateway selected as the bundle forwarder. However, it suffices as a first-cut solution because we do not anticipate much inter-region user mobility in our system. If it proves necessary, we shall refine the solution in the future.

2.5.3 On the use of distributed consensus

In our design, the gateways of a region need to achieve a distributed consensus in various situations. We enlist each of them below.

1. When a bundle reaches the gateways in the source region, they need to decide on who will send the bundle further, to reduce the usage of the gateways' bottleneck connections to the Internet. As described in Section 2.3 the gateway getting the bundle first informs all other gateways that it will be sending the bundle to its destination region. Since this *claim* is sent over the Internet it is received by all other gateways of the region immediately, i.e. within a few seconds. Consensus requirement comes in to the picture when two or more gateways send the *claim* for the same bundle at the same time. It can be seen that the probability of such a situation arising is very low. Further, even if all the gateways who *claimed* the bundle together send the bundle, it only results in poor utilization of the Internet links of the gateways but does not create any form of incorrectness in the system.

2. During the registration process if the node was registered to another region earlier a *COA* needs to be sent to the old region. To minimize usage of the bottleneck links only one of the gateways should send the *COA* bundle to the old region. Similar to the point above the gateway getting the *REG* bundle first shall *claim* to send a

COA bundle to the old region. In case two or more gateways *claim* at the same time (again which is expected to happen rarely), two or more gateways in the old region will receive the *COA* bundle. This will result in two or more gateways send in the bundles destined to the registering node to the new region. Again this only leads to poor utilization of the Internet links of the gateways.

3. During the registration process the gateway that makes the first entry in the DHT for the registering node creates a new record for that node in the DHT. All other gateways update this entry by adding its own address to the list of gateways in the record. However, DHT does not provide anyway of making this update atomic. Hence, only one gateway should update the entry at a time. Given n gateways wanting to update the same entry at any point of time they must agree upon which gateway updates the entry first. This requires achieving distributed consensus. However, in the DHT a record is identified by the <key,value> pair instead of just the key. This means that there can be two records in the DHT with the same key and different values. We use this flexibility to avoid the requirement of distributed consensus. When two gateways, say G1 and G2, want to edit the same entry, they will both read the record and add its own address to the local copy of the record. When they both add their copies of the record both the records will be stored separately in the DHT even when they have the same key. Later when a gateway wanting to send a bundle to this node will query the DHT with the key derived from the destination address, both the records will be returned. This approach leads to a little bit of redundancy but avoids the need to achieve distributed consensus.

Thus we have shown that even not achieving strict agreement between all of the gateways may only affect the performance of the system and not its correctness. Further, we have also shown that such degradation of performance is expected to be relatively rare.

2.5.4 Caching

We use caching at the gateways for keeping temporary records of various kinds of information to optimize the system performance. We describe below each of the scenarios where caching is used.

1. When a gateway has a bundle to be sent to a destination present in another region it performs a DHT lookup to retrieve the list of gateways of the destination region. There is a high probability that this gateway will have a few more bundles to send to the same destination in the near future. In this case caching the DHT entry of a node can prove to be useful. This reduces the overhead of performing DHT lookup frequently. It also reduces the usage of slow Internet links of the gateways. However, if a node whose DHT is cached by a gateway changes its region then the cached entry must be expired. This is done when the gateway, say G1, uses its entry to send a bundle to one of the gateways, say G2, of the old region. At this point G2 sends a *cache expired* message to G1 which then flushes the DHT entry and makes a fresh DHT lookup.

2. During the registration process, when a gateway receives a *REG* bundle, it records the address of the registering node. In future, if a bundle arrives at the gateway through the intra-region routing process, it can check if the destination address is present in the list of addresses recorded during the registration process. In that case the gateway can identify that the destination belongs to the same region and it can avoid making a DHT lookup. Again such caching reduces the usage of the bottleneck links connecting the gateways to the Internet. However, when a *COA* bundle is received by a gateway it needs to inform all other gateways of the region about the same so that all the gateways remove the address of the node for which *COA* has been received from the list of address belonging to its region.

2.6 Implementation Overview

We have extended the DTN reference implementation [1] for our architecture because the DTN reference implementation provides us with a basic framework for working in an intermittently connected environment. The default routing mechanism in DTN reference implementation uses a routing table to route bundles to the next hop. However, a routing table may not contain all the links that the node has. For example consider a node A is connected to two nodes B and C through two different network interfaces. Further, C is also connected to B resulting in a triangular topology. The routing algorithms running on each of the three nodes use a metric such that the value of the metric for the link $A - B$ is higher than the sum of the metric values of the links $A - C$ and $C - B$. Assuming lesser metric value is better, the routing table at A will consist of entries for B and C with link $A - C$ as the out going link. Thus the information regarding the link $A - B$ is lost from the routing table. Instead, a topology table that simply lists all the neighbors suffices to flood bundles. The topology table that we use also indicates the kind of node (kiosk, gateway or bus) at the other end. This information is used by gateways to flood the bundle only on the link which has a bus or a kiosk at the other end, restricting the flood within the region.

We have made a few simplifications in the first-cut implementation. First, we do not use any mechanism to achieve a distributed consensus. As explained earlier this only affects the performance of the system and not its correctness. Next, when a gateway has to send a bundle to another region, after looking up the corresponding DHT entry, the gateway randomly selects a destination gateway from the list. This prevents the need to have bus schedule information. Further, we have implemented only G2G inter-region routing solution with random gateway selection. Hence, as required in the G2G solution, during the registration process all the gateways create/update the entry corresponding to the registering node at the DHT.

Chapter 3

Evaluation and Analysis

In this chapter we first study the performance of the intra-region routing algorithm from various dimensions through a combination of experiments and simulations. We study how ME and DC, both independently and together, improve the performance of the flooding mechanism. We identify three potential bottlenecks in the system which may affect the performance of the flooding mechanism. Based on these studies we estimate the amount of data a node may be allowed to generate without the system getting overburdened.

Next, we study, through simulation and analysis, the load that a proxy may experience in the $G2P$ and $G2PB$ solutions to inter-region routing. Through the same analysis we also study the load a centralized scheduler may experience under the $G2GC$ solution. Through this study we re-establish the intuition that we developed and presented in Table 2.1.

3.1 Evaluation of Flooding Mechanism

3.1.1 Evaluation setup

To study the effect of ME and DC on the flooding mechanism ran a simulation, written in **C**, consisting of 100 nodes representing a *region* of the deployment. With a deployment scale of $100,000$ nodes, we believe division of the system into 1000 regions with 100 nodes per region is an appropriate breakup. Of these 100 nodes 10 are designated as buses and 10 other as gateways. This translates to 8 kiosks connected by one bus. We show that this is a reasonable assumption below.

Before we describe the simulation environment any further it is important to describe a typical bus schedule for a bus that visits villages. Our inquires at the Kanpur rural bus station has revealed that a bus that travels to rural regions typically takes off early in the morning from a city. It reaches a large town in about 2-3 hours visiting about 10-12 villages on the way. It breaks for about half hour before visiting several other villages, going on to reach another large town. The bus then ferries between these two large towns till the end of the day. All the villages in between these two large town are a visited multiple times during a day. At the end of the day it either returns to the city or stays back in one of the large towns. We can think of a couple of kiosks existing between two large towns and between a large town and a city, and gateways existing at each large town and the city. Hence, it is reasonable to have one bus connecting about 8 kiosks and to have 10 gateways per 80 kiosks. Next, we describe bus schedules in our simulations below.

We model our bus schedules on the typical bus schedule described above, which consists of two large towns, a city and several villages in between each. Let the buses be numbered $B_1....B_{10}$ and the gateways be numbered $G_1....G_{10}$. Each bus B_i starts in the morning from the gateway G_i. All the buses follow the schedule described above. However, only one of the two large towns is assumed to have a gateway. The second gateway that a bus B_i visits is G_{i+1}. Thus only two adjacently numbered buses (except for B_1 and B_{10}) have a two hop connection between them. This one node that connects two buses through this two hop connection is a gateway. For example, the node that forms a two hop connection between B_1 and B_2 is G_1. A kiosk has only one link, that is to the bus that visits it. Each bus visits 8 kiosks. Thus, if we number the kiosks from K_1 to K_{80} then bus B_i visits kiosks $K_{8*(i-1)+1}$ to $K_{8(i-1)+8}$. When a bus visits a kiosk or a gateway a *contact* is said to be established and data transfer can take place.

We simulated the above described setup with two different bundle generation rates $1bundle/min$ and $0.5bundle/min$ at each node. The bundle generation rates effectively define the load that will be borne by the system; for example $1bundle/min$ translates to $100bundles/min/region$ and $100,000bundles/min$ in the whole system. If we assume bundle size to be $50KB$ (the maximum DTN reference implementation currently supports), then the bundle generation rate translates to $50KB/min$ and $25KB/min$. Thus $1bundle/min$ with bundle size of $50KB$ is equivalent to close to $1.2GB/month$. This capacity, we believe, is extremely high for applications like e-mail and e-governance, and we would like to see how the system performs with these capacities. Since no measurement study provides with the distribution of size of bundles (or even IP packets) that are generated from a rural kiosk we do not use any particular bundle size. Rather we perform the study measuring number bundles in the simulation. This also gives us the flexibility of plugging in an appropriate bundle size value at a later stage to arrive at more concrete conclusions or to make better informed predictions regarding performance of the system.

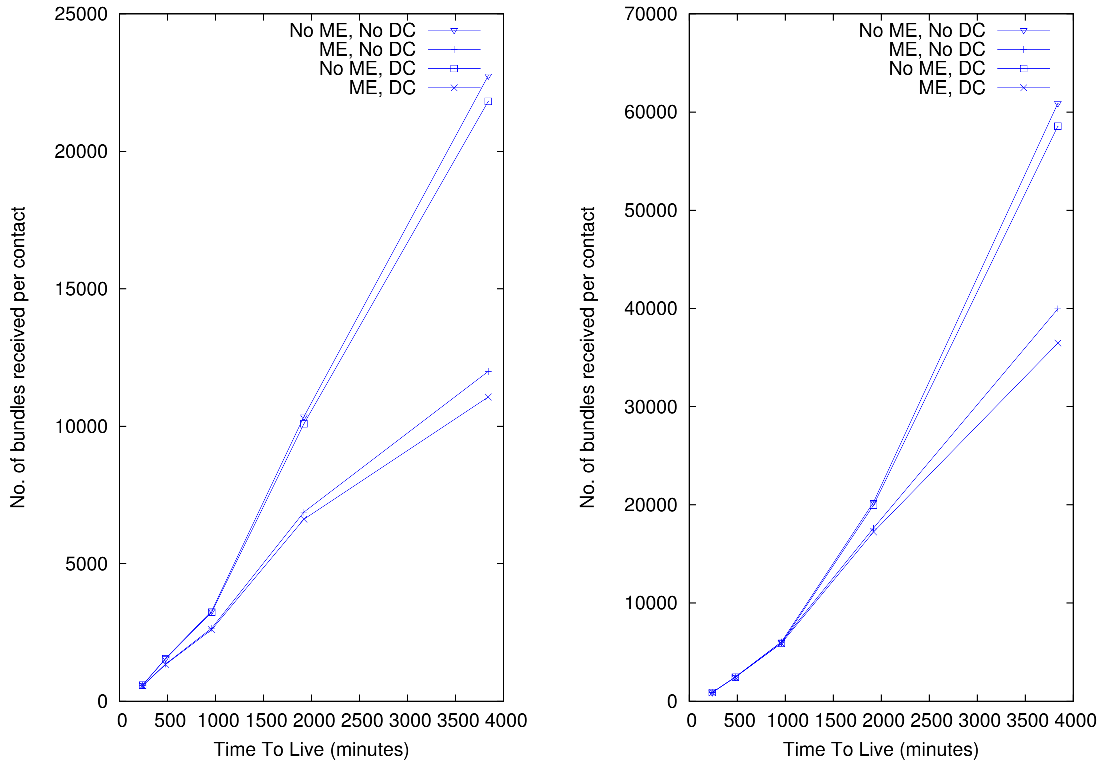
We vary time-to-live (TTL) value, the time a bundle lives from its creation time, for the bundles and run the simulation for each value of TTL. The TTL values used are 4hrs (240 minutes), 8hrs (480 minutes), 16hrs (960 minutes), 32hrs (1920 minutes), and 64hrs (3840 minutes). Thus the simulations consist of four tuning parameters: (a) TTL values for bundles, (b) bundle generation rates, (c) whether ME is present, and (d) whether DC are generated.

We study two metrics in the simulation: (a) average/maximum number of bundles in store during the simulation and (b) average/maximum number of bundles received by a node during all the contacts. The avg/max bundles in store metric gives us an idea of the number of bundles a node may have to store in order to participate in flooding. Number of bundles received per contact can give us an indication of the amount of data transfer is required per contact. We can then plug in a bundle size to obtain the above metrics in terms of bytes. The simulation is run for 60 hours. The results of the simulation are shown below.

3.1.2 Evaluation results

Figures 3.1 and 3.2 show the number of bundles received per contact by a node. For average the value is averaged across all the nodes and for maximum we have extracted maximum number of bundles received across all the contacts across all the nodes. The foremost point of observation is that the number of bundles received per contact increases with increase in TTL values for bundles. This is expected as with increased life time a bundle survives longer in the system. As a result the bundle is a part of larger number of contacts and hence adds to the metric depicted in the figures.

Secondly, we clearly see a huge difference between the curves where ME is present and the curves where ME is not present. In the graph showing maximum number of bundles exchanged per contact, we see that the curves without ME rise exponentially with TTL

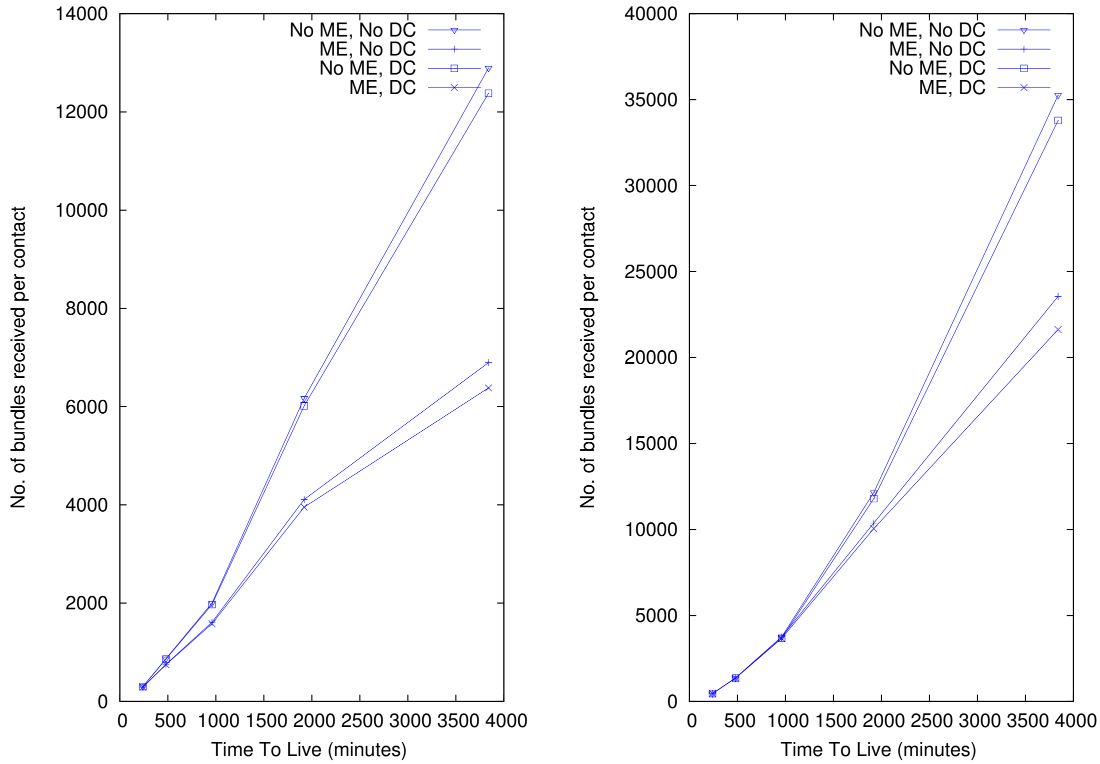


(a) Average (b) Maximum

Figure 3.1: Average and maximum number of bundles received per contact across all the nodes. Bundle generation rate = $1 bundle/min$

value while the curves with with ME rise only linearly. Further, the number of bundles received per contact when ME is used is close to half of the number of bundles received when ME is not used. This result clearly points out the fact that almost half of the bundles received by a node during the contact are duplicates. Thus ME saves wastage of network bandwidth significantly.

We also see a slight difference between the curves where DCs are generated and the curves where DCs are not generated. Note here that when DCs are not used the bundles that have already reached their destinations are still being exchanged; but when DCs are used the bundles get discarded as and when their corresponding DC spreads across the nodes. Here we have not counted DCs themselves as bundles to bring out the difference between DC and $NO - DC$. For any particular TTL value and choice of ME (whether in use or not), the difference between number of bundles received without DC and number of bundles received with DC give the number of death certificates received per contact. Notice



(a) Average (b) Maximum

Figure 3.2: Average and maximum number of bundles received per contact across all the nodes. Bundle generation rate = $0.5 bundle/min$

that the number of death certificates received per contact is close to 1% of total number of bundles received. Based on this observation one may conclude that DC are not very effective in reduction of network bandwidth wastage. However, we believe that the poor results shown by DC is because the network that we in our simulation was sparsely connected. Because the network was sparsely connected the bundles timed-out before reaching their destinations. If they did not time-out then their death certificates timed-out soon after they were generated as by design death certificates expire at the same time as their corresponding bundles.

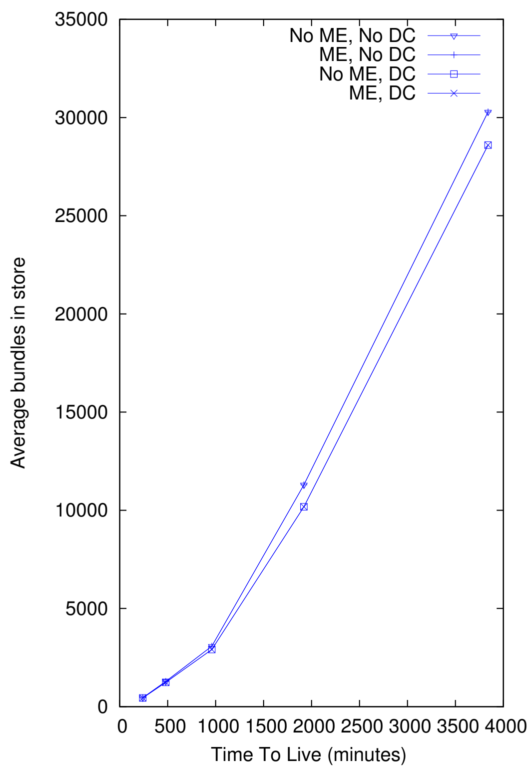
Bundle Generation Rate	Average Bundles	Average DC	Max Bundles	Max DC
$0.5 bundle/min$	15589	1389	43964	5040
$1 bundle/min$	25690	2425	76501	8597

Table 3.1: Average and maximum number of bundles and death certificates received per contact for a network with *dense* topology. TTL = $1920 minutes$. ME was used.

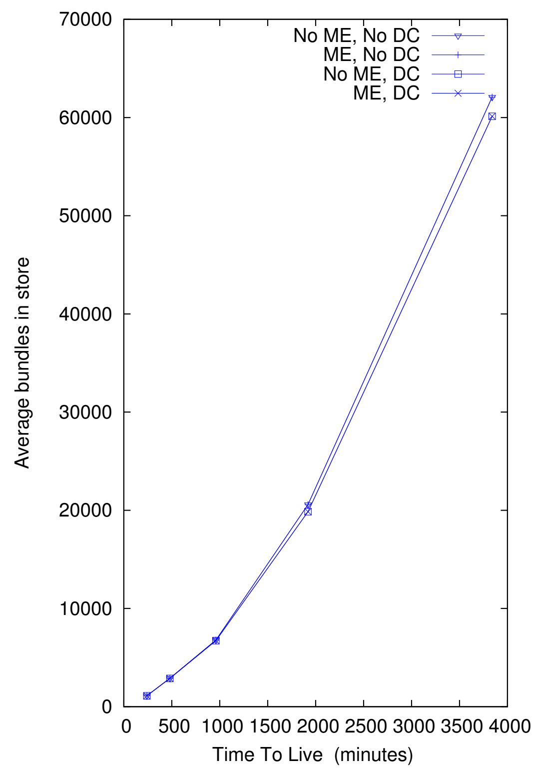
To corroborate the above conjecture we conducted a simulation with two different bundle generation rates for a network that was more densely connected. In the new network a bus visited four gateways instead of two. Further, we also increased the number of times a bus visited a gateway. The results of the simulation are shown in Table 3.1. We can see that with enough connectivity available, death certificates for nearly 10% of the total bundles that were received. Thus it is clear that sparse connectivity in the earlier simulation caused DC to have lesser impact than expected.

It is important to mention here that the metric observed in Figures 3.1 and 3.2 is the number of bundles received by a node per contact. This metric is different from number of bundles exchanged per contact. However, the two can be related when dealing with averages as

$$numberofbundlesexchangedpercontact = 2 * numberofbundlesreceivedpercontact$$

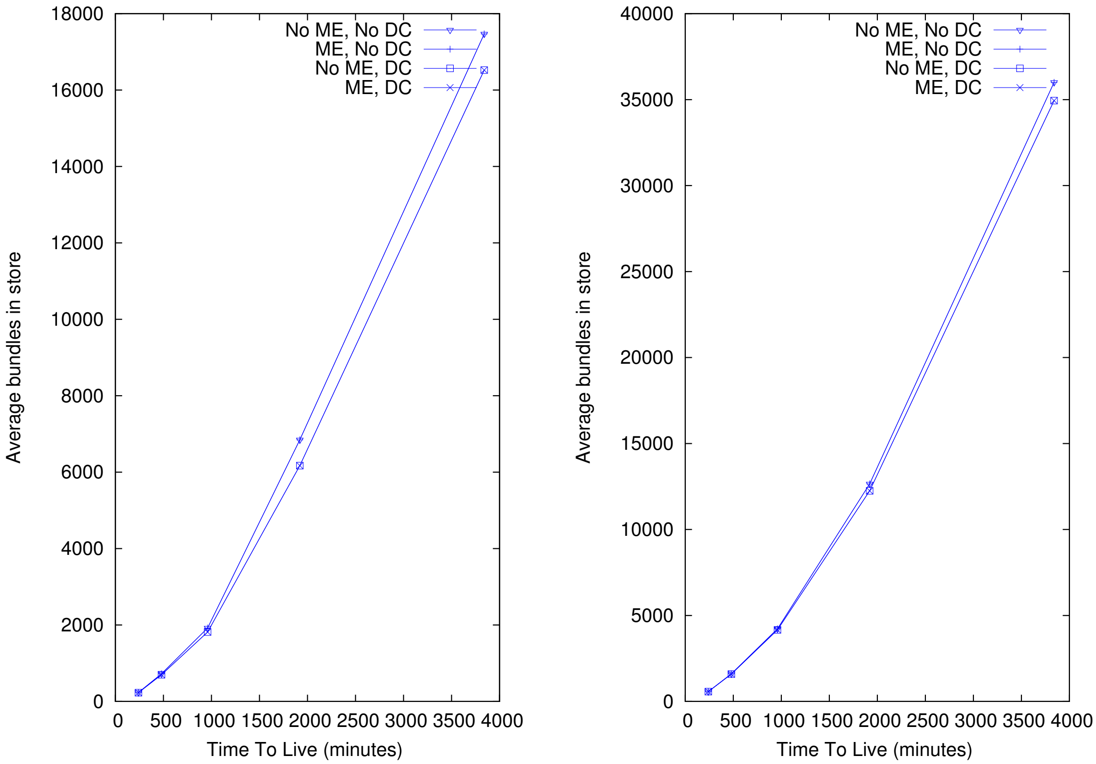


(a) Average



(b) Maximum

Figure 3.3: Average and maximum number of bundles in store across all the nodes. Bundle generation rate = $1 bundle/min$



(a) Average (b) Maximum

Figure 3.4: Average and maximum number of bundles in store across all the nodes. Bundle generation rate = $0.5 bundle/min$

We now study the results related to number of bundles in store. Figures 3.3 and 3.4 show the number of bundles in store on the nodes during the simulation. For calculating the average at each node the average is recalculated after ever ten insertions/deletions from the store. This value is then averaged across all the nodes. Similarly, for calculating maximum value we keep track of the maximum number of bundles in store. After every ten insertions/deletions the max value is recalculated by comparing it with current number of bundles in store. We see from the figures that the number of bundles in store increase dramatically with increase in TTL values. Although this is expected, it is not very clear whether the number bundles in store increase linearly or exponentially with increase in TTL values. The figures show no difference between curves with ME and curves without ME. This is because ME only helps in avoiding transfer of duplicate bundles over the air. Even without ME duplicate bundles are discarded from the store. So the use of ME is does not make any difference to the number of bundles in store. We also notice that DC helps reduce storage only to a little extent. However, by the same arguments made earlier, we believe this is due to the nature of connectivity considered in our simulation. We expect DC to make a much greater impact in a densely connected network.

3.1.3 Analysis of potential bottlenecks

Next we identify and analyze various aspects of the problem setting that are potential bottlenecks for system performance. Note that this work is aimed at deployment of a large scale network of kiosks in rural regions. The problem setting makes it imperative that we consider the cost of deployment during the design of the system itself. Given the cost constraints low cost, lower power single board computers are an appropriate choice for routers to be set up on the buses. Such single board cost about $200 in the market. We assume here that kiosks located at villages and gateways present in towns/cities are general purpose computers with decent CPU and disk capacity as they are used by people for various services. With this design choice in mind we now analyze the potential bottlenecks that may cause difficulties in the implementation of the scalable routing architecture. We identify three potential bottlenecks:

1. Non-volatile storage

2. Network bandwidth/contact time

3. CPU capacity of single board computers

Non-volatile storage

Non-volatile storage like hard disk drives are cheaply available. Currently a 40GB drive can be bought for less than $50 in the market. We expect to have hard disk drives with capacities of at least tens of GBs at mobile routers as well as kiosks and gateways. Considering this capacity available at all the nodes we now calculate the maximum storage space required as indicated by our simulations. For any node and any TTL value the maximum bundles in store is about $62,000$ as shown in Figure 3.3. Even if we considered the bundle size as $50KB$ the storage space required would be close to $3.1GB$, which is far lesser than the disk storage capacity available at any of the nodes. Hence, we can safely conclude that in any real deployment non-volatile storage should not be source of performance bottleneck.

Network bandwidth/contact time

The amount of data transfered during a contact depends on the amount of time the contact stays up and the network bandwidth. Since a contact between a bus and a kiosk remains established only during the time the node on the bus is in the *range* of the kiosk, we estimate one minute as a reasonable contact duration. In Section 2.2 we estimated that about 75MB of data can be transfered in one minute using an 802.11g connection. Based on this if we choose 16 hours or 960 minutes as the time to live value, with ME in use and bundle generation rate of $1bundle/min$ (figure 3.1) about $2500 * 2 = 5000$ bundles are transfer per contact on average. Using this information we can conclude that average size of the bundles should be around $75000/5000 = 15KB$ if bundle generation rate is $1bundle/min$. This effectively translates to $21.6MB$ per day per node or close to $650MB$ per month per node. This bundle size is less than 50KB size allowed by DTN. Thus the network bandwidth does form a performance bottleneck for our system. However, we believe, the upload capacity is still much more than required by the applications like e-mail and e-governance that are under consideration.

CPU capacity of single board computers

Although theoretically the network capacity provides enough upload capacity per node it is important to find out if the single board computers are fast enough to operate at such high data rates. We analyzed this aspect of the system through a set of two experiments. We ran the scalable routing solution, implemented by us, to transfer data between a single board computer and a laptop over the wireless network. Such a set up was chosen because we expect the gateways and the kiosks to be regular PCs with high CPU capacities and mobile routers present on the buses to be low power single board computers. The single board computer was a Soekris [4] 4521 box with AMD Geode 133MHz processor and 64MB RAM. The soekris box used compact flash $CF2.0$ card with write speed of $50X$ or $7.5MBps$ and capacity of $1GB$. Thus along with the operating system running from the flash card the card was also used for storing bundles.

In the first experiment we transfered $45MB$ of data over an 802.11b wireless link, operating at $11Mbps$, from the soekris box to the laptop. In the second experiment we transfered the same amount in the same direction, but over an 802.11g wireless link operating at

$54Mbps$. After the transfer of every $5MB$ we noted the time taken for the transfer. The bundle size was fixed to be $50KB$. The first experiment was run 8 times and the second 5 times. The time values obtained were averaged over all the runs of the experiment. The results of the experiments are displayed in Table 3.2. Note that although ME was used, the time taken by the metadata exchange process is not reported. This is because the time taken by ME is not the focus of the experiment.

802.11 Type	5MB	10MB	15MB	20MB	25MB	30MB	35MB	40MB	45MB
b	11.85	25.86	39.56	53.58	67.40	81.70	96.59	112.41	127.92
g	10.08	22.01	32.19	42.78	54.41	66.34	77.97	91.00	104.16

Table 3.2: Average time to transfer chunks of $5MB$ of data. Bundle size = $50KB$. All time values are in seconds.

The experiment results show that using 802.11g wireless link close to $30MB$ can be transfered in one minute. This is less than half of our theoretical estimate of $75MB$ made above. To confirm that this decrease in performance is due to low CPU capacity we can see that although the physical data rate of 802.11g link is almost 5 times that of 802.11b link, the application layer throughput obtained by 802.11g link is less than twice that of 802.11b link. We now recalculate the amount of upload the system can allow with the newly measured data rate. With $30MB$ instead of $75MB$ transfered in one minute, choosing 16 hours as the TTL value average bundle size should be $6KB$ with data generation rate of $1bundle/min$. This translates to $8.64MB$ per day per node or $260MB$ per month per node. We believe that with this amount of upload one can provide common services like e-governance, e-mail, and other special services like aAqua [17] satisfactorily. Clearly, CPU capacity of the single board computers form the primary bottleneck of our solution, constraining the capacity of the system to a much larger extent than network bandwidth.

3.2 Analysis of Inter-Region Routing Solutions

In Section 2.3 we compared the various inter-region routing solutions. The comparison was based on simple intuition. We had claimed in that section that in G2P and G2PB the proxy may become a performance bottleneck. Further, we also mentioned that in G2GC the centralized scheduler may pose even more severe performance issues. Here we analyze to what extent the proxies and the centralized scheduler can become a bottleneck.

We assume the same bundle generation rate of $1 bundle/min$ as in our simulations. Further, we assume that a region consists of 100 nodes. This amounts to $100 bundles/min/region$. Globally this means 100,000 bundles are generated every minute. If we assume the worst case that for each bundle a request is send to the centralized scheduler, then the centralized scheduler will need to process 100,000 requests per minute. this translates to 1,666 requests per second. Thus the calculations show that the number of requests per minute for the centralized scheduler are not very large. There are two points to note here. First is that the centralized scheduler does not lie along the data path. This can significantly reduce the performance requirement of the scheduler. Second point to note is that the centralized scheduler is expected to decide to which gateway of a destination region should a gateway send the bundle it wants to send. This means that processing a request at the scheduler can more complicated than performing a simple look up in a table. Thus, whether the centralized scheduler becomes a performance bottleneck depends on the performance of algorithm that processes each request. To optimize the performance the centralized scheduler may be deployed on a cluster of high performance machines similar to modern web servers.

For the proxies in G2P and G2PB the load is much lesser. Assuming all the 100,000 bundles are equally divided to be sent to all the regions, the proxy of each region will have to process 100 requests every minute; that is 1.66 requests per second. At these number of requests to be processed every second, we believe, the scheduling shall not be a cause of bottleneck. However, it should be noted that in G2P and G2PB, the proxies lie along the data path; i.e. each bundle destined to a region goes through the proxy of that region. If use the same numbers as above then. Each proxy will have to send 1.66 bundles every second. This, again, is not a very difficult task considering the bundle size of $6KB$ obtained in Section 3.1.3.

In Section 2.3 we mentioned that a simple way of selection of a destination gateway is to choose one gateway at random. We argued that this may lead to imbalance among the gateways. We now develop an insight into how much imbalance can happen in this random selection approach. Let us again consider 100,000 bundles being generated every minute in the system. If they are equally divided among all the regions, then 100 bundles will arrive at each region. Let us assume that there are 10 gateways per region. This means that for each of the 100 bundles one gateway is selected at random from the 10 gateways and the bundle is sent to that gateway. To observe how much imbalance can be caused through random destination gateway selection we performed a simulation.

In the simulation we kept track of the *overload* for each gateway. Every minute 100 bundles were generated and each of them were assigned to one of the ten gateways. In a perfectly balanced system we would expect $100/10 = 10$ bundles assigned to each gateway. At the end of the assignment if a gateway was assigned n bundles more than 10 then its load would be increased by n. On the other hand if the gateways was assigned n bundles less than ten, then its load would be decreased by n. Note, however, that the load of a gateway was never made negative. Thus if a gateway was assigned 8 bundles and its current load was 1 then its load would be set to 0 and not -1. The *overload* of a gateway effectively represents utilization of the bottleneck link of the gateway. Each run lasted 60 hours, during which 360,000 bundles were generated and assigned to the 10 gateways. The simulation was run 10 times. The maximum value of *overload* observed for any gateway during all the 10 runs was 564. If we translate it to number of bytes with the assumption of $6KB$ per bundle, it amounts to $3.3MB$ or $27Mb$. For a 100kbps line this is equivalent to 4.5 minutes worth of data which is not a lot considering that the simulation was run for 60 hours. Thus the imbalance caused by random gateway selection does not cause any significant imbalance to the system. However, the above arguments hold only till the bundle size is $6KB$. Note that this figure of $6KB$ has been arrived at after studying the performance bottlenecks in Section 3.1.3.

3.3 Evaluation Summary

Thus we have evaluated various aspects of our design. The evaluation brings to fore following salient points.

1. The processing capacity of low power single board computers form the primary bottleneck of our system.

2. For a network size of 100,000 nodes, TTL value of $16hrs$, and 100 nodes per region the solution architecture can support up to $260MB$ of upload per month per node for the given problem setting.

3. Imbalance caused by random selection of destination gateways during inter-region routing does create catastrophic problems when total bundles generated in the system are equally divided among all the regions and then sent to their gateways.

4. Use of proxies in G2P and G2PB is not expected to create performance issues.

5. The centralized scheduler in G2GC may become a bottleneck based on the kind of scheduling used for destination gateways selection.

Chapter 4

Related Work

DTN in general is relatively unexplored field. A first attempt at dealing with the problem of routing in DTN was made only as late as in 2004 [12]. Hence, most issues in DTN routing remain largely untouched. There have been only two attempts at solving the problem of routing in mechanical backhaul networks [19] [11]. However, one of them focused primarily at the complete architecture while the other only considered a partial component of the problem. We touch upon various works that deal with routing in DTN below.

Jain *et.al.* [12] were the first to look at the problem of routing in delay tolerant networks in general. They considered four different cases, each with decreasing amount of information available for making routing decisions. The information considered by them were (a) aggregate summary of contacts that happen in the network, (b) information regarding exact time and duration for each contact in the network, (c) queuing information at all the nodes, and (d) traffic demand at all the nodes. They studied various algorithms with varying amount of availability of information to conclude that clever routing algorithms with only partial information available can perform satisfactorily. In particular, *Minimum Expected Delay (MED)* algorithm was shown to use only summary of contacts and yet gave good results in terms of delay and delivery ratio. However, this algorithm did not describe any practical way of estimating the contacts summary.

Jones *et.al.* [13] designed the $MinimumEstimatedExpectedDelay(MEED)$ algorithm which could be practically implemented. The *MEED* algorithm is a link state style algorithm where the path metric used is a variation of contacts summary. This metric is calculated by observing the contact up and down times of a node and maintaining a sliding window history of the same.

It is important to point out, however, that the routing protocols above are all generally applicable to delay tolerant networks. The problem setting considered by us have peculiar characteristics like use of low power, high secondary storage routing modules and availability of low speed Internet connections at the gateways. The above mentioned algorithms are not suitable for those characteristics. Seth *et.al.* [19] [18] proposed an architecture for mechanical backhaul networks. They also proposed a routing mechanism, which we briefly describe below.

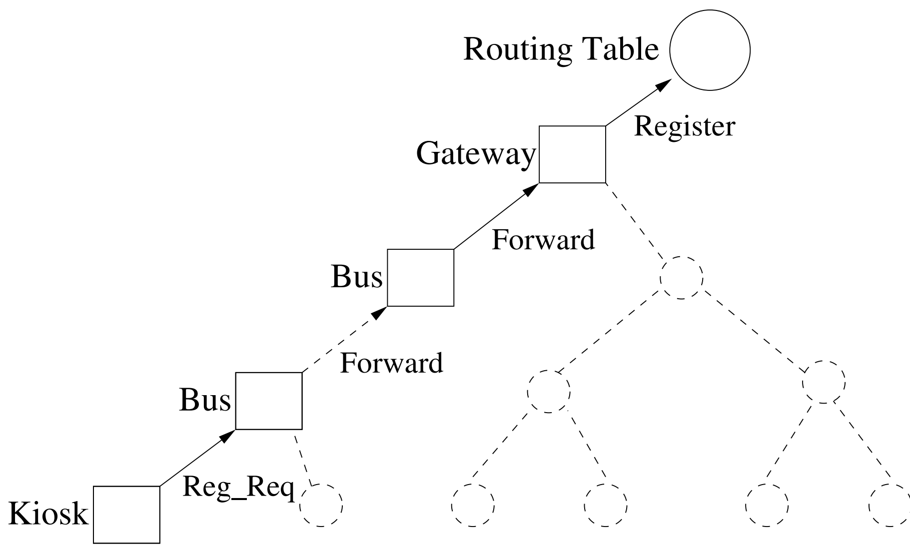


Figure 4.1: Registration process in Mechanical Backhaul Networks Architecture.

[19] combines location management and routing in the same process. In this solution, each router is configured with a default route that marks the interface used to communicate 'with the Internet'. In order to declare its location a node sends a *Register* message along the default route. Each node forwards this message along its default route up to the gateway. The gateway, being connected to the Internet, makes an entry for the node in a globally accessible routing table with itself as the next hop for that node. This process of registration results in a sink tree as shown in Fig. 4.1.

[19] combines this registration process with reverse path formation. When a router receives a registration message from a particular endpoint ID on, say, interface i, then it installs a route for that endpoint going through interface i. This reverse path is used for delivering data to the node. To communicate with a node, a node simply puts the node's

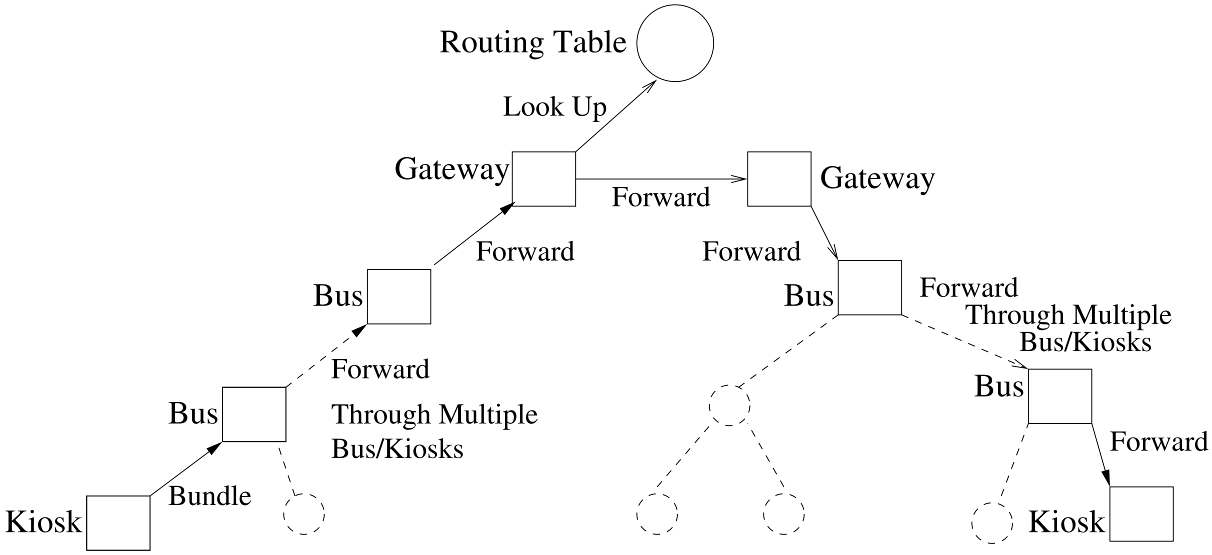


Figure 4.2: Schematic diagram of deployment.

address in the destination field. This is forwarded along the default route to a gateway, which looks up the global routing table to find a peer gateway, and thence the reverse path to the node. Fig. 4.2 shows this process of reverse path forwarding.

Although reverse path forwarding is scalable, it is brittle: a single link failure can potentially isolate a large number of nodes from the network. This is even more true when the scale of deployment considered is as large as ours. Our work focuses on achieving scalability along with high bundle delivery probability. Further, our design takes into consideration the aspects peculiar to rural deployment setting and analyzes the design based on the low cost requirements of our problem domain.

Guo *et.al.* [11] considered the problem of sending data from one *region* to another in a mechanical backhaul network where a *proxy* acted as a point of contact between two regions. The solution described by them, however, did not consider the scale of deployment in their design. The part of the algorithm which delivers data from a proxy to the destination node in its region is relevant to us and so we describe it in brief here. In [11] the proxy maintains a queue for each kiosk present in its region. When a unit of data arrives at the proxy to be delivered to kiosk K_i it stores the data in queue i. The problem now can be defined as given n gateways and m queues, how do we schedule a unit of data from queue i on gateway j such that overall delay is reduced. Note that in mechanical backhaul networks the primary component of delay is the delay for a ferry to reach the gateway first to collect data and then reaching the destination kiosk. Guo *et.al.* propose to use bus schedules for solving the scheduling problem. The essential idea is that if a bus B_i going to destination kiosk K_j is just about to reach the gateway G_k then data destined for K_j should be sent to G_k. As mentioned in Section 2.3, we propose to use this scheduling mechanism as a part of the inter-region routing solution. Hence, the work in [11] is complementary to ours.

There have thus been two attempts at dealing with routing in mechanical backhaul networks, but none of them suit our settings. In our work, we have designed a routing architecture for mechanical backhaul networks that scales to the order of 100,000 nodes and is robust, providing high bundle delivery probability. The architecture takes into account the low cost requirements and low bandwidth availability and utilizes the resources well. Finally, our work also provides a basic analysis of the system to draw some initial conclusions on the performance of such a large scale deployment.

Chapter 5

Conclusion and Future Work

If this work we have designed, implemented and evaluated a routing solution for mechanical backhaul networks that scales well to the order of country wide networks and provides high delivery probability through replication. We have achieved scalability by creating a hierarchy in the deployment and have attained robustness through flooding. Through a combination of simulations and a prototype implementation we have successfully shown that flooding is in fact a good design decision. We have found that for the considered problem setting, with a network deployment of 100,000 nodes, one node can be allowed up to $260MB$ of upload per month. This upload capacity is more than satisfactory for most applications currently exististing in rural regions; for e.g. e-mail, e-governance services, transfer of agricultural knowledge to the rural regions and so on. We have also identified that in our problem setting it is the low power single board computers that form the bottleneck of the system. Any improvement in these computers shall result in the improvement of the capacity of the system in terms of maximum upload allowed per node.

With these results at hand we envision the following action items as future work:

1. Implementation of G2P and G2PB: Since the analyses show very light load on the proxies, its implementation and use is feasible.

2. Stress test the implementation for deployment: Although the scalable routing solution has been implemented it need to to be tested for long durations to make the implementation robust and bug free.

3. Deployment of the scalable routing architecture: A deployment on a large scale of a few hundred nodes can provide us with valuable information about traffic patterns

and system usage. The routing algorithm can benefit for this information and to optimize its performance.

Bibliography

[1] Delay Tolerant Networking Research Group. `http://www.dtnrg.org`.

[2] IEEE P802.11, The Working Group for Wireless LANs. `http://grouper.ieee.org/groups/802/11/`.

[3] Mission 2007 - Every village a knowledge centre. `http://www.mission2007.org`.

[4] Soekris Engineering. `http://www.soekris.com`.

[5] M. Best and C. Maclay. Community Internet Access in Rural Areas: Solving the Economic Sustainability Puzzle. In *Centre for International Development (CID)*, 2002.

[6] C. Blattman and R. Jensen and R. Roman. Assessing the need and potential of community networking for developing countries: A case study from India. Technical report, February 2002.

[7] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss. Delay tolerant network architecture. In *Internet Draft*, September 2006.

[8] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *PODC '87: Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, pages 1–12, New York, NY, USA, 1987. ACM Press.

[9] E. Brewer and M. Demmer and B. Du and M. Ho and M. Kam and S. Nedevschi and J. Pal and R. Patra and S. Surana and K. Fall. The case for technology in developing regions. In *IEEE Computer*, May 2005.

[10] K. Fall. A Delay-Tolerant Network for challenged Internets. In *SIGCOMM*, 2003.

[11] S. Guo, M. Gadheri, A. Seth, and S. Keshav. Opportunistic scheduling in Ferry-Based Networks. In *WNEPT*, 2006.

[12] S. Jain, K. Fall, and R. Patra. Routing in a delay tolerant network. In *SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 145–158, New York, NY, USA, 2004. ACM Press.

[13] E. P. C. Jones, L. Li, and P. A. S. Ward. Practical routing in delay-tolerant networks. In *WDTN '05: Proceeding of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, pages 237–243, New York, NY, USA, 2005. ACM Press.

[14] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. In *ASPLOS, San Jose, CA*, Oct. 2002.

[15] J. Ott and D. Kutscher. Drive-Thru Internet: IEEE 802.11b for automobile users. In *INFOCOM*, 2004.

[16] A. Pentland, R. Fletcher, and A. Hasson. Daknet: Rethinking Connectivity in Developing Nations. In *IEEE Computer*, 2004.

[17] S. Sahni and K. Ramamritham. Delay Tolerant Applications for Low Bandwidth and Intermittently Connected Users: the aAQUA Experience. 2007. Poster paper.

[18] A. Seth, P. Darragh, S. Liang, Y. Lin, and S. Keshav. An Architecture for Tetherless Communication. 2005. Technical Report.

[19] A. Seth, D. Kroeker, M. Zaharia, S. Guo, and S. Keshav. Low-cost Communication for Rural Internet Kiosk Using Mechanical Backhaul. In *MOBICOM*, September 2006.