Chapter 3

Mobile Ad Hoc Networks: Rapidly Deployable Emergency Communications

Niaz Chowdhury and Stefan Weber

Contents

This chapter describes Beatha, a transport protocol for rapidly deployable emergency communications. The Irish Gaelic word *beatha* means "life." In keeping with the meaning of its name, the proposed protocol introduces a lifelike quality to end-to-end data delivery in emergency response, where nodes are not the first point of communication but rather their services are. Beatha makes use of characteristic-based network architecture to route packets from source to destination and along its network layer counterpart forms a new network paradigm that replaces the Internet Protocol–based network in wireless environment.

3.1 Introduction

Over the past two decades, researchers across the globe have extensively investigated wireless communications. The mobile ad hoc network (MANET) is one of the outcomes of that research. The community has defined the MANET in many ways:

- A MANET is a collection of wireless mobile nodes that cooperatively form a network without infrastructure [1].
- A MANET is a dynamically reconfigurable wireless network that does not have a fixed infrastructure [2].
- MANETs consist of a group of mobile and autonomous nodes that are directly interconnected to each other. These networks are independent of any communication infrastructure such as access points or base station and rely entirely on node cooperation for relaying information from data source to intended destination [3].
- A MANET is a group of mobile wireless nodes that cooperatively form a network independent of any fixed infrastructure or centralized administration [4].
- A MANET is a network that works under dynamic routing with a multihopping mechanism and has no centralized body to govern the network under which the network has to communicate [5].

From the above definitions, it is clear that for the most part nodes in MANETs are wireless and mobile. The network is dynamically reconfigurable, does not have a fixed infrastructure, and relies on node cooperation for delivering data from one

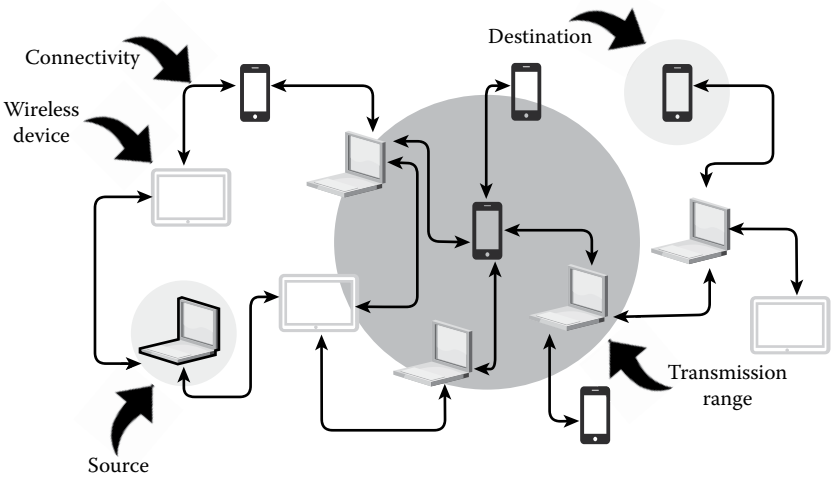

Figure 3.1 Architecture of mobile ad hoc network.

node to another (Figure 3.1). Although IEEE 802.11 is the most frequently used link layer protocol for MANETs, other protocols such as IEEE 802.15.4 (ZigBee) or IEEE 802.16 (WiMAX) standards can also be used as underlying medium access control (MAC) and physical layer protocol [6].

The infrastructureless nature and being able to operate without central administration control make MANETs very effective in playing an increasingly important role in extending the coverage of traditional wireless networks. This technology is widely used to deploy decentralized communication in remote places [6]. There are many applications that can be built around MANETs. Military operations and emergency police response are already in use. Vehicle-to-vehicle communication is another dimension derived from this network that helps to prevent accidents and make communication available in next-generation vehicles. Nevertheless, in recent years emergency communication for disaster recovery has become one of the most demanding applications where MANETs can be best fitted [7].

Communication over wireless networks, however, is currently managed at higher layers based on the use of Internet Protocol (IP) and its transport layer counterpart Transmission Control Protocol (TCP). Wired networks assume that nodes are interconnected by immobile elements and individual streams of communication do not interfere with each other. However, a rising number of ubiquitous systems with wireless communication functionality removes the basis for these assumptions, and the nature of communication between nodes in a given area becomes inherently sequential in order to avoid the interference of concurrent transmissions. This has led to poor performance of transport protocols

used in wired networks in their wireless MANET counterparts. Another prominent difference between wired and wireless networks is the change in topology. Topology seldom changes on wired networks. Arguably, the only reason for altering the topology of a wired network is to have a router down. However, in wireless networks, especially in MANETs, a number of causes bring changes to the topology. Because MANETs are comprised of mobile nodes with differing velocities, the chance that a node will move away is very high. Moreover, in MANETs each node acts as a router and because of its decentralized nature a node may join or leave at any time. This feature, however, is also responsible for creating changes in the topology. These frequent changes eventually cause two problems that largely hamper end-to-end communication in MANETs, particularly in emergency situations where nodes move frequently. First, they cause frequent link failures while routing data from the source to the destination and, second, a node serving a particular task may move away, resulting in a permanent service failure.

It is therefore eminent that protocols designed for wired networks are not adequate to serve the purposes of wireless networks, and the necessity for an alternative approach to an IP network model is at its peak now. In recent years, several approaches have been proposed from different parts of the world. Some proposals have come in the form of cross-layer implementation, whereas some proposals have dealt with multipath routing to enhance transport layer performance. However, initiatives that replace both IP and TCP are still rare, and reliable communication over such an alternative paradigm has not yet been established.

This chapter stresses two main points: first, the importance of switching to an alternative paradigm in order to emphasize service over provider and, second, the need for a more suitable transport layer capable of adjusting the delay in emergency communications. Both motivations have their roots in the drawbacks that occur when protocols designed for wired networks operate in a dynamic wireless environment.

The justification for switching to an alternative paradigm lies in the nature of MANETs. Because MANETs are highly mobile and nodes change positions frequently, a fixed identifier like an IP address is not useful in a wireless environment. For example, a disaster recovery mission, where rescuers search for victims in a relatively broad area, might have a situation where someone requires a digger. In such a case, that person would be happy to have the service available for use; it would not be necessary to know the person providing the service. As a result, tracking nodes with a fixed identifier in these applications often do not play an important role. This example makes it clear that in emergency communication MANET services should take priority over service providers. A provider may remain visible or invisible but should not be taken into account for communication purposes, and the identity of nodes needs to be revealed based on the services they offer. In recent years, a number of network platforms have been proposed that keep this principle in mind. Content centric networks [8], attribute-based routing [9], and Publish–Subscribe Architecture [10] are a few examples that provide alternative communication based on data or services.

The need for a more suitable transport layer is particularly important because in emergency situations, networks face long delays and sometimes might get disconnected for awhile. Because the basic assumptions of existing transport protocols designed for wired networks do not include properties found in wireless networks, they exhibit poor performance while operating over an IP network. The design of the new transport protocol takes three main issues into account:

1. Congestion can be the principal reason for data loss in wired network but in wireless ad hoc networks mobility or link failure could be other reasons for this problem.
2. Low-bandwidth wireless networks, in contrast to wired networks that have dedicated bandwidth, are not suitable for window-based flow control.
3. Because of the collision avoidance feature of medium access control protocols, segment-by-segment acknowledgment is an obstacle for achieving desired throughput.

3.2 Rapidly Deployable Emergency Communications

The nature of emergency communications is different. This kind of communication is rapidly deployed in the event of a natural disaster, accident, war, or similar situation, where using an infrastructure-based network is difficult or sometimes impossible. It is evident from the type of scenarios that the users and the data they exchange are also very different from regular data being exchanged on the Internet.

Let us consider a real-life scenario as illustrated in Figure 3.2. A disaster recovery team has been trying to rescue victims at a site. This team would possibly be comprised of rescuers, diggers, emergency medical technicians (EMTs), an ambulance, firefighters, a base camp, gateway, and so on. These entities need to communicate with each other to support the rescue mission as a team. For example, a rescuer might seek assistance from an EMT to give immediate medical support or a digger might report on the state of a victim by sending photographs. Again, diggers or ambulances might need to communicate to decide where to provide service at any given moment. One noticeable attribute common to all communication in emergency response is service dependency. A rescuer is not concerned with the person serving as a digger, EMT, or firefighter as long as he or she is receiving the service requested. Thus communication occurs between services instead of nodes.

The existing architecture, however, does not support this concept. At the inception of the Internet, the IP-based communication model became the *de facto* standard for establishing communication at higher layers. Wireless networks, which are the backbone of establishing emergency communications, require their own protocols at the transport and network layers to provide adequate performance in the described scenarios. Although medium access control in wireless network is

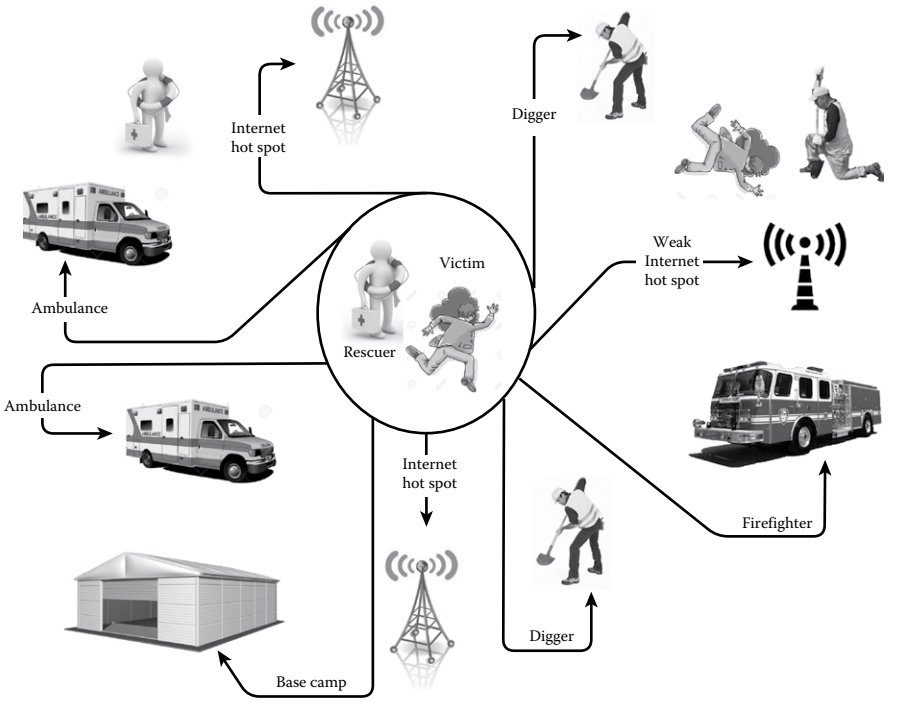

Figure 3.2 Disaster recovery program.

managed by specialist protocols, for example IEEE 802.11, higher layers continue to be managed by IP protocols. Despite the fact that this old practice has been the cause of poor performance at both the network and transport layers, until recently sufficient initiatives have not been undertaken to explore alternative approaches.

Over the past two decades, a number of redesigns have been proposed for the end-to-end approach in distributed systems [11] and conventional Internet structure [12]. Nevertheless, it was only when Wi-Fi became popular that researchers from around the world started undertaking initiatives to design alternative protocols to IP. Content-Based Networking [13] and Content-Based Routing [8] were two of those early initiatives. In recent years two other protocols, Adaptive Attribute-Based Routing [9] and Publish–Subscribe Architecture [10], have been proposed. These protocols, however, are designed for application-specific network layers, except content-based routing, which has some degree of functionality to provide unreliable transport layer support, but they are not a successful alternative to the TCP/IP paradigm. One of our main challenges in this work was to find a suitable routing protocol that works with Beatha set up on top of it. After exploring the existing protocols in the literature, we ended up choosing Uisce, which is an anonymous routing protocol and replaces IP in the communication protocol stack.

3.3 Alternative Paradigm

Because of the highly mobile nature of MANETs, nodes change their positions frequently. This behavior suggests that a fixed identifier such as an IP address does not have a great impact on controlling the network and nodes need to keep changing their service directory. Let us take a more technical look at the example described in the previous section. Figure 3.3 shows that Node 3 is a rescuer with multiple nodes around it. At the network layer, only their IP addresses are visible. Suppose this rescuer finds a victim and wants to communicate with an ambulance, ask for help from a digger, or seek Internet access. In order to do that, he must know which node is providing what services—that is, he needs to go through a service discovery procedure (Figure 3.4).

This can be achieved in two possible ways. The interested node (in this example, Node 3, the rescuer) may keep track of the IP addresses of every neighboring node and maintain a table of services associated with each IP address. While looking for a specific service, it can just look up the table and get the IP address of the desired node. Unfortunately, this solution is unrealistic due to the uncertainty of node movement in MANETs. An alternative approach can also be used. The interested node may go to every neighboring node and ask whether it is an ambulance or a digger or such. This second approach will significantly increase overhead in the network.

This example points out one important aspect: IP addresses may not be very helpful in a highly mobile environment, and therefore a network protocol is

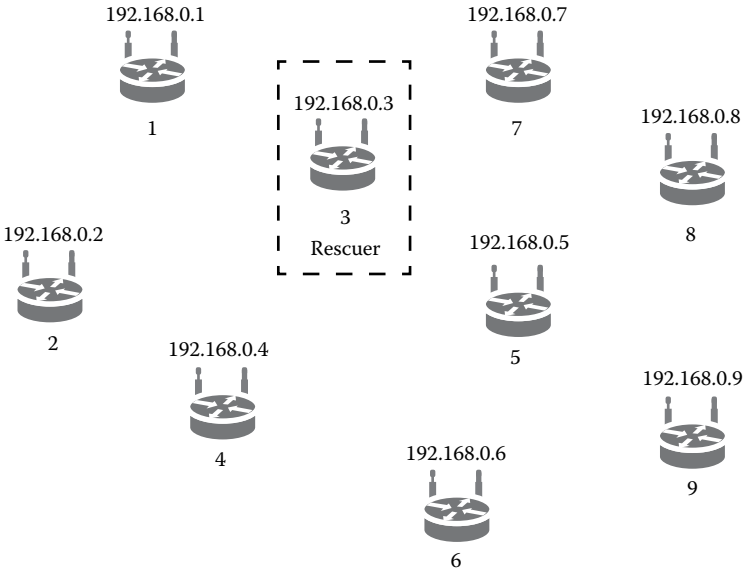

Figure 3.3 Network prior to service discovery.

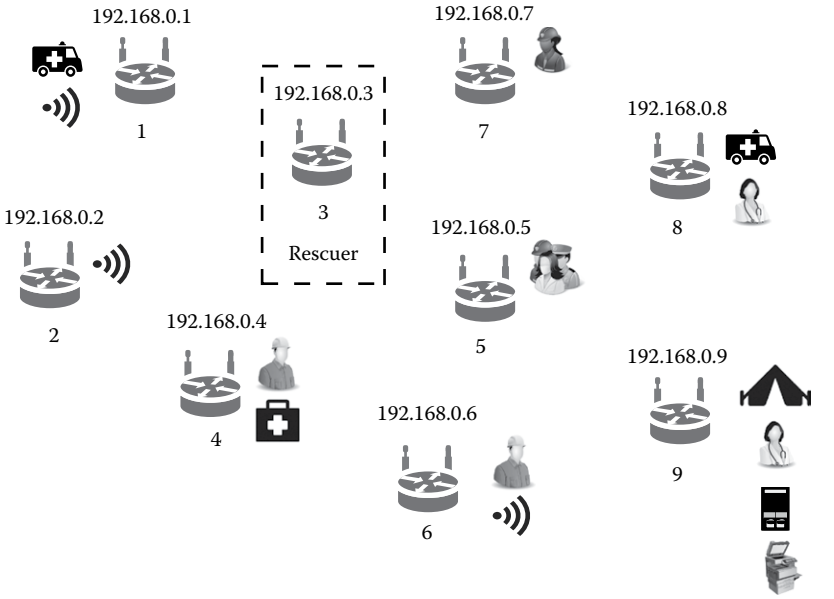

Figure 3.4 Network after service discovery.

required that omits the concept of addressing nodes with fixed identifiers and finds routes based on the services available at each node. As a result, the demand for an alternative approach was intense and Uisce grabbed attention by meeting that need.

Uisce is a characteristics-based routing protocol for MANETs designed and developed by Guoxian and Weber [14]. In this protocol, the term *characteristic* refers to a service. From that point of view, Uisce is very similar to other service-oriented routing protocols. However, unlike IP routing protocols it does not identify nodes with a fixed identifier. It even cannot differentiate nodes from one another. It only communicates based on characteristics, overlooking nodes that retain that characteristic. Because of this special property Uisce is called an *anonymous routing protocol*. Note that this chapter does not try to improve the performance of Uisce; rather the proposed transport protocol works on top of it with a view to forming an alternative paradigm.

The real-world analogy used for characteristic dissemination in Uisce is a natural river. The flow of water in a river is driven by gradients in the landscape, from high to low altitudes, and multiple rivers merge at a confluence to form a new river, which flows on with a combined water level. Uisce associates a characteristic with a potential value representing the water level of the river. At the source node of a characteristic, the potential describes the capacity of the characteristic. The dissemination of characteristic information is performed through broadcasting. A host node broadcasts its characteristic information periodically. It helps neighboring nodes to keep their knowledge up to date. One of the most significant issues in

Uisce is that a node only has information about the characteristics of its one-hop neighbors; the rest of the network is anonymous to it.

Uisce operates based on three functions and a parameter to control the potential changes in values during the propagation. A cost function takes the current potential as input and calculates a reduction of the potential. For each characteristic, all nodes in a network use a uniform specification of this function. A cost compensation function is applied to reduce the potential value cost along the hop. It provides individual nodes with the flexibility to affect the characteristic flow that runs through them. Finally, the third function, known as the *fusion function*, calculates the potential of multiple flows merging into one. In addition to these functions, Uisce has a control parameter called w_{dif}. A node that contains a characteristic with potential w accepts an incoming flow of the same characteristic only if the incoming potential is not lower than $(w - w_{dif})$.

The specifications of these functions may vary from characteristic to characteristic. In order to have them operate correctly, a condition is given that confirms the stability of the characteristic and precludes any reverse flow. Uisce sets the maximum value of the cost compensation function to c_{max} and the minimum value for a characteristic to run further as *threshold*. Hence, the condition stands as follows:

$$w_{dif} + c_{max} < threshold \tag{3.1}$$

In short, the higher the capacity value of a characteristic, the further it may propagate and the closer to a characteristic source, the higher capacity could be sensed. To achieve an optimal path, data messages in Uisce are forwarded towards the direction of potential gradient that balances route selection between the distance and the source node.

Uisce has an on-demand and temporary characteristic known as the *trace characteristic* that is created by the source node. This temporary characteristic enables a return path between destination and source for each data packet. This special characteristic is, however, allowed to propagate only within a limited region and its lifetime is also very short. Despite the fact that Uisce is an autonomous routing protocol, the proposed transport protocol will use this characteristic to send acknowledgment by making the communication bidirectional.

3.4 Challenges

The design aspect of a transport protocol for a rapidly deployable mobile environment involves many challenges. This involves (if not limited to) sequential access to the wireless medium, controlling the flow of data, and the consequences occurring due to link failure. These challenges often prevent the transport protocol from obtaining the best possible performance. Nevertheless, a trade-off between

challenges and performance can be made if potential handles are observed, studied, and addressed during the design phase of the protocol.

Wireless networks are different compared to wired networks when they communicate with multiple stations simultaneously. For example, when multiple stations in the Ethernet try to access the medium at the same time, the MAC of the transmitting station identifies the collision using a collision detection mechanism and switches to a retransmission phase based on an exponential random back-off algorithm. Although this technique is very effective in a wired LAN, it is not a suitable solution for wireless. This is mainly for two reasons. First, implementing a collision detection mechanism would require implementation of a full duplex radio, capable of transmitting and receiving at the same time. This approach would increase the price significantly. Second, in a wireless environment, we cannot assume that all stations can hear each other, which is the basic assumption of the collision detection scheme. In wireless, when a station is willing to transmit and senses that the medium is free, it does not necessarily mean that the medium is free around the receiver area. In order to overcome these two problems, the 802.11 standard, instead of using a collision detection mechanism, utilizes a different approach known as *collision avoidance* together with a positive acknowledgment scheme. This collision avoidance inherently makes all communications sequential [15].

Figure 3.5 explains this scenario with more context. It shows a four-hop path with five wireless nodes and stations placed in such a way that there is a fixed distance between each of them. In this scenario, when Node A tries to access Node B, Node B cannot access its neighbor. Even Node C needs to remain idle to avoid interference. At this given time, only Node D can access its neighbor Node E or vice versa. This sequential nature of the wireless medium eventually creates dependencies between nodes and decreases accessibility of the whole path at a given time. Most transport protocols currently used in MANETs are built for wired networks and do not account for this sequential nature of 802.11 MAC. In a multihop network, such transport protocols inappropriately interpret delivery delay caused by this sequential medium access and trigger remedies that are meant for something else.

This medium access problem is also responsible for obstructing reverse flow on the path. Because acknowledgment is an integrated part of any reliable transport protocol, naturally a flow gets created opposite to the flow of the data packet. Most transport protocols including TCP have segment-by-segment acknowledgment, and for each individual or group of segments the receiver node sends a message acknowledging successful delivery of the respective segments. This acknowledgment message creates a reverse flow. As described earlier, the wireless medium is

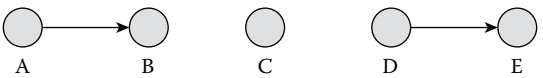

Figure 3.5 Sequential access to the wireless medium.

sequential in nature; while accessing nodes, this reverse flow becomes an obstacle for the forward flow and vice versa.

The process of identifying congestion in the network is also an important task for any reliable transport protocol. Often segments get lost in the network due to congestion, and transport protocols need to retransmit the missing segments. In the course of this process, some segments that are assumed lost may arrive at a later time. TCP usually suffers from this kind of problem when routing at the network layer is performed over a long multihop path in a wireless network. This delayed arrival of segments is interpreted by TCP as out-of-order data and it resends acknowledgment for the last ordered segment. The TCP receiver node continues to do this every time it receives such a delayed segment and, in response, the TCP sender node moves into the fast retransmission phase after receiving at least three acknowledgments. This special phase of TCP flow and congestion control assumes that the segment that immediately followed the delayed segment was also lost and therefore retransmits it without waiting for the timers to expire. This mechanism is useful in wired networks, as it helps maintain the flow even if the network is congested. However, in a wireless network that is not the case, and this misinterpretation unnecessarily triggers congestion control.

Moreover, round trip time (RTT) and retransmission time-out (RTO) in TCP are designed based on assumptions obtained from wired networks. However, in a wireless environment an unexpected situation often occurs and deceives TCP into calculating RTT and RTO inappropriately. We will revisit some of these challenges in the next section and see how they affect TCP flow control and end-to-end throughput if they are not properly taken care of.

3.5 TCP Flow Control

TCP, the most common transport protocol, is not suitable for a rapidly deployable wireless environment in emergency rescue missions mainly for two reasons. First, we have already seen that TCP is designed based on assumptions obtained from wired networks, and it therefore fails to achieve the performance it usually demonstrates in wired networks in a wireless environment. Second, applications that are used in rescue missions are different and largely differ from the kind used in everyday life. As a result of these combined factors, TCP flow control often triggers inappropriate actions that result in poor performance. In this section, we are going to review those inappropriate behaviors exhibited by TCP and pave the path for a potential solution to be presented in the next section.

TCP follows window-based flow and congestion control mechanisms [16,17]. Its flow control is closely tied to its congestion control and operates in a synchronized fashion. One of the blunt decisions these flow and congestion controls make is to identify any delay as congestion. Although this assumption is correct for wired networks, in a wireless environment delay could occur for a number of reasons,

including link failure, temporary isolation, rapid movement, poor signal, and so on. TCP is completely unaware of these situations and triggers congestion control as soon as it identifies any delay [18].

TCP congestion control is divided into four phases: *slow start*, *congestion avoidance*, *fast retransmit*, and *fast recovery*. Among these four phases, slow start and congestion avoidance are responsible for slowing down data flow. Slow start is the initial phase of any TCP connection—it slowly probes the network and starts increasing the pace of the data flow. It holds two variables, namely congestion window (*cwnd*) and slow start threshold (*ssthresh*). When a new connection is established, TCP sets *cwnd* to one segment size (a segment size is either the size announced by the other end or 512 bytes by default) and *ssthresh* to 65,535 bytes. Every time an acknowledgment (ACK) is received, the *cwnd* is increased by one segment. This mechanism limits the sender to transmitting up to the minimum of the *cwnd* and advertised window (*rwnd*). On receiving each ACK, the TCP sender increases *cwnd* once, creating a provision for sending more data next time. This process, in fact, gives slow start an exponential breakthrough over RTT.

Slow start continues until *cwnd* is less than or equal to *ssthresh*; otherwise, it gets phased out and TCP flow control enters into congestion avoidance. Every time an ACK is received, this phase increases *cwnd* by $segsize^2/cwnd$, where *segsize* is the segment size. Thus *cwnd* grows linearly compared to slow start, which grows exponentially. During this phase, the increment of *cwnd* gets fixed by a maximum of one segment regardless of the number of ACKs received.

In event of a time-out, TCP assumes that congestion has occurred and invokes congestion control by resetting the *cwnd* size to one segment and triggering slow start again. Nevertheless, this time slow start continues until TCP reaches half of the size of the window when congestion occurred and then it swiftly moves to congestion avoidance. During this phase, TCP window growth will be much more conservative and, if no losses are detected, *cwnd* will grow no more than one segment per round trip. As TCP performs these phases assuming that network is congested, it unnecessarily slows down the data flow in event of link failure, temporary isolation, and so on and requires a longer period to get back to normal mode, even if new paths are established or nodes get out of temporary isolation quickly. This observation suggests that window-based flow control is not suitable for wireless networks. Link failure or situations involving temporary isolation can separate one or more nodes from other parts of the network for a specific period. The transport protocol needs to know when such events occur and when nodes are getting back to normal business. Rate-based flow control can be more effective in performing that job.

3.6 Overview of Beatha

Beatha is designed to overcome the challenges that transport protocols encounter in a rapidly deployable wireless environment. The primary objective of this protocol

is to provide reliable end-to-end transport layer support for applications used in disaster recovery and rescue operations over characteristics-based anonymous communication. The following subsections describe components of Beatha in brief.

3.6.1 Connection Management

Transport protocols with reliable support do not communicate only on an end-to-end basis. They operate in a process-to-process manner through connection-oriented communication for specific streams. Although the use of Uisce benefits Beatha in many ways, in turn it creates a new problem for Beatha. Facilitating an anonymous underlying routing protocol to work with a connection-oriented continuous data stream seems difficult. Because Uisce is only able to see nodes within a one-hop distance, it is blind to the network and can recognize only its immediate neighbors.

Figure 3.6 gives more insight into this matter by illustrating an abstract model of how packets are seen from Uisce's point of view. It shows Node A transferring two sets of data, represented by the white and the grey packets, while Node C is transferring just one set, represented by the black packets. In this example, a set of data indicates a stream that may simply contain a file of multiple segments. Although segments from two different files are handed over to Uisce at Node A, it cannot identify those segments separately. To make things worse, it also cannot report which segment belongs to who while delivering those segments. For example, Node F receives packets from both A and C that Beatha receives in random order from Uisce.

This problem can easily be solved in TCP/IP with the help of creating a socket that involves an IP address and a port number. This combination is unique for any stream at a particular time and helps keep track of segments of the same stream. In the proposed alternative paradigm, however, the characteristic is the only tracing

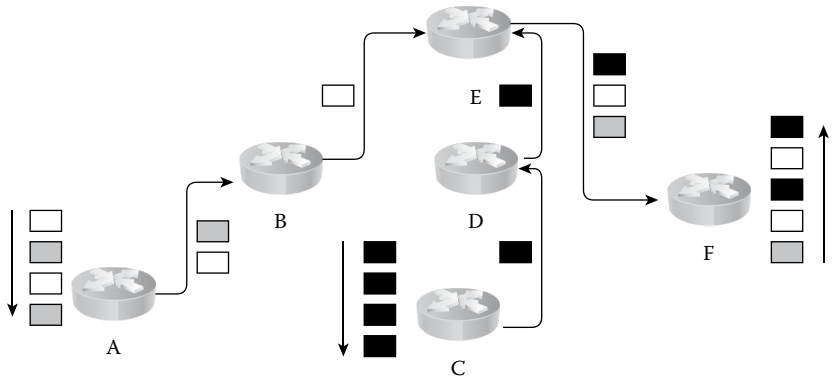

Figure 3.6 Packets from Uisce's point of view.

element to discover a node from the network. In order to take advantage of this in a wireless environment, earlier we deliberately gave up the uniqueness of any tracing element that makes it possible to have the same characteristics in multiple nodes. Therefore, a transport protocol operating over anonymous networks cannot correlate the concept of IP address–based connection management and requires a different mechanism to distinguish nodes and their streams from each other.

Beatha connection management consists of three elements: connection frame (CF), active connection table (ACT), and active reception table (ART). Each connection is associated with a set of specific variables that controls the procedure. Amongst those variables, members of CF act as the identification property of each stream. CF is comprised of a unique combination of three different variables: requested characteristics (RC), source tracker (ST), and destination tracker (DT), where RC is the characteristics requested by the application, ST is a randomly generated number for the source, and DT is another randomly generated number for the destination. Note that ST and DT are generated by the source and destination nodes, respectively.

In Beatha, ACT and ART are two tables maintained by the sender and receiver nodes, respectively. The ACT contains the CF of all initiated requests, whereas the ART contains the CF of all accepted requests. The entries in these two tables are finite and the maximum number of connections that can be initiated and accepted is controlled by a connection threshold called C_{thresh}. When Beatha receives a communication request from its application layer, it generates a CF and creates an entry in the ACT but leaves the DT field empty. It then sends a connection request with RC. Uisce, the underlying network protocol, is responsible for delivering this request to a node that holds the requested characteristic. When the node on the other end receives this request, it accepts it by generating a connection acceptance message with a DT. It also creates an ART entry for this connection. A connection is said to be established when the source Beatha receives a completed CF from the receiver and acknowledges it with a connection establishment message.

Once a connection gets established between two nodes, they enclose a CF within each data or control message to differentiate communication streams from each other. Despite the fact that the underlying network layer is anonymous, because of the presence of this connection frame, a receiver node can explicitly identify segments of appropriate streams.

3.6.2 Segment Management

Conventionally, transport protocol receives a data stream from the application layer and chunks them into small pieces called *segments*. These segments are the smallest unit of data at the transport layer. Each segment contains the necessary transport protocol header in front of its payload. When the transport layer passes its segments to the network layer, those segments are then encapsulated into a packet and routed to other nodes. This work maintains this convention of data flow at different layers.

For each connection, Beatha has a data buffer to hold its segments and keep track of associated variables. There are two types of buffers: the source node, which is responsible for initiating connections and sending data, maintains the outgoing segment buffer, and the receiver node, which acts as the destination, maintains the incoming segment buffer (ISB).

3.6.3 Acknowledgment

Beatha is designed especially to avoid reverse flow in data communication. Unlike TCP, Beatha does not have a segment-by-segment acknowledgment scheme. In fact, it never sends an acknowledgment to the sender in order to ensure successful delivery. It only sends an acknowledgment when it does not receive a segment. Specific to Beatha, this is called *negative acknowledgment* (NAK).

In Beatha, segments are organized in chronological order and the sender node sends them sequentially. It helps the receiver node to understand the arrival process more effectively. When a segment arrives, if Beatha finds that one or more segments have a smaller sequence number than the most recently received segment but have not arrived yet, it starts individual timers for all of them and waits for a specific period. If the timer expires but those segments are still due to arrive, the receiver node sends a NAK requesting the sender to retransmit missing segments.

This acknowledgment procedure works based on three variables that are located in the ISB control information. These variables are *isReceived*, *isSequenced*, and *isTimerSet*. Beatha initializes these variables with *false* at the time of establishing the connection. Subsequently, the values of these variables get updated and help the sender maintain the retransmission process smoothly. Briefly, when a receiver node receives a segment Beatha checks three things. First, it checks whether the segment has been received or not. If not, then it marks the associated *isReceived* variable *true* and checks whether the immediately previous segment is sequenced—that is, whether all the segments above the immediately previous segment have been received. If that segment is sequenced, Beatha marks *isSequenced* of the received segment *true*. However, if the immediately previous segment is not sequenced, there are two possible situations: either that segment has not been received yet or at least one segment above is yet to arrive. So Beatha starts checking *isTimerSet* for all the segments upwards that have not yet been received. If any such segment is found for which *isTimerSet* is false, Beatha starts a timer for it. If the respective segment does not arrive before the timeout, the receiver node sends a NAK to the sender to retransmit the segment and sets another timer for it. The details of these two timers are discussed later.

3.6.4 Retransmission Timer Management

Unlike the general convention for retransmission timer management, which is located at the sender's end, the retransmission timer in Beatha is controlled from the receiver's end. As described in Section 3.6.3, when the receiver experiences

a time-out of an expected segment, it sends a NAK to the sender, requesting it to send the segment again. The receiver's end maintains a timer for this NAK and sends another NAK if it does not receive the data segment before the time-out. Note that Beatha also uses the same timer when it waits for an expected segment.

Most transport protocols including TCP use Jacobson's algorithm to calculate RTT and retransmission time-out (RTO). This algorithm was first introduced for TCP in the Request for Comments (RFC) 2581 [16]. However, later calculation of RTT and subsequently RTO was modified through an updated version of this algorithm described in RFC 2988 [17]. This new version introduced RTT variance instead of a fixed RTT calculation.

As per this algorithm, once the first RTT measurement (say, R) is made, the host calculates two variables as follows:

$$\text{RTT} = R \tag{3.2}$$

$$\text{RTTVAR} = \frac{R}{2} \tag{3.3}$$

The subsequent RTT measurement will then maintain the following equations:

$$\text{RTTVAR}_{\text{new}} = (1-\beta) \times \text{RTTVAR}_{\text{old}} + \beta \times |\text{RTT}_{\text{old}} - R| \tag{3.4}$$

$$\text{RTT}_{\text{new}} = (1-\alpha) \times \text{RTT}_{\text{old}} + \alpha \times R \tag{3.5}$$

In the above equations, R is the original RTT of the last segment, RTT is the measured round trip time, and RTTVAR is the RTT variance. α and β are two constants, defined as 1/8 and 1/4, respectively.

However, Jacobson's algorithm proposed for TCP involves operations based on feedback from previously travelled segments. As Beatha does not send regular acknowledgments, it is not possible to use this algorithm directly. Nevertheless, in order to adapt this algorithm to Beatha some necessary modifications are introduced in this chapter.

Beatha requires a timer based on the RTT when it sends NAKs for missing segments and waits for retransmissions from the sender's end. One of the difficulties in the RTT calculation in Beatha is its unconventional acknowledgment approach. Because the receiver never acknowledges any segments, it is not possible to get an RTT from data segments. Hence, RTT in Beatha is not calculated based on a real round trip; instead, it maintains a virtual return time to incorporate Jacobson's algorithm within it. This calculation is performed as follows: the Beatha sender includes a time stamp in the header of each segment it sends to the receiver's end. On receiving a segment, the receiver extracts this time stamp and deducts it from the current time to get the travel time (say *tt*) from the sender's to the receiver's end. Soon after, it calculates R_v using the following formula to replace R in Jacobson's algorithm.

$$R_v = 2.5 \times tt \tag{3.6}$$

There is a justification behind calculating R_v in this way. We saw earlier that Beatha segments receive a privilege when they access the medium. They do not face a reverse flow created by acknowledgments. However, when a NAK passes through a long multihop path, it faces a delay due to the constant flow of data segments. In order to adjust this delay, the segment travel time is multiplied 2.5 times instead of just doubling it.

Hence, by replacing R by R_v in Equations 3.4 and 3.5, we get

$$\text{RTTVAR}_{\text{new}} = (1-\beta) \times \text{RTTVAR}_{\text{old}} + \beta \times |\text{RTT}_{\text{old}} - R_v| \tag{3.7}$$

$$\text{RTT}_{\text{new}} = (1-\alpha) \times \text{RTT}_{\text{old}} + \alpha \times R_v \tag{3.8}$$

The RTT obtained from the above equations is used in Beatha to calculate the RTO. In this process, the first RTT is straightforward R_v and will be calculated just after receiving the connection request. Once the connection is established, Beatha starts calculating the RTT based on Equations 3.7 and 3.8.

Earlier, we saw that Beatha sets a timer on two different occasions: it sets a timer for expected segments and also sets a timer after sending NAKs. Although these timers are set for two different reasons, one single retransmission timer is sufficient to serve the purpose. The following expression as a function of RTT describes RTO calculation in Beatha.

$$\text{RTO} = fx(\text{RTT}) \tag{3.9}$$

where

$$fx(\text{RTT}) = 1, \text{ if } \text{RTT} < 1.0$$

$$fx(\text{RTT}) = \text{RTT} + 4 \times \text{RTTVAR}, \text{ if } RTT \geq 1.0$$

The calculation is carried out in this manner in order to adjust the unpredicted delay that the segment may face due to numerous reasons produced by the wireless environment. Sometimes it happens that a segment or group of segments face delay due to link failure or temporary isolation. However, this does not indicate that the segments are lost. During the design phase of Beatha, it was observed that an RTO having more than a 1-second period allows the network enough time to find a new route. Nevertheless, when the RTO is very small, it results in a time-out. This time-out triggers a retransmission request for a segment or group of segments that have not yet been lost. In order to reduce this kind of duplicate transmission, Beatha sets a 1-second minimum threshold for its retransmission time-out.

3.6.5 Flow Control

The flow control is a process of managing the data transfer rate between the sender and the receiver. The amount of data a sender can send at a given time towards a particular receiver is decided by the flow control scheme of the sender. The flow control plays an important role in a rapidly deployable wireless network, as it needs to change the data rate on the fly. Unlike window-based TCP flow control, Beatha opted out of rate-based flow control because it helps establish more command over the data transfer.

A transport protocol views the network as a black box and sends segments to lower layers to deliver them to the other end. The general assumption of sending a group of segments to lower layers significantly differs in wireless networks compared to a wired network. In a wired network, when a transport protocol tries to send multiple segments to the receiver's end, it assumes that they will take almost equal time to reach the destination from the source node. However, we have already seen that in a wireless network, the medium can only provide adequate support for the first segment and the rest face an increasing delay to get access to the medium. It eventually delays their round trip and gives a false interpretation to the transport layer. When the transport layer assumes that the previously passed segment should have been delivered by this time and provides another bunch of segments, it creates buffer overflow at the MAC layer and results in a time-out for multiple segments. In order to cope with this problem, Beatha uses flow control that is completely unrelated to acknowledgment and follows its adaptive nature to adjust its rate with the change of network topologies.

This flow control process (Figure 3.7) sends two different types of data blocks. The first block is called *complete block* (c_b). The length of this block is constant. Each complete block is again divided into small partial blocks (p_b). Beatha sends a whole p_b at a time and waits for a period of delay known as *internal delay* (d_{int}) and sends another p_b with a view to eventually sending a whole c_b. In between two c_b Beatha

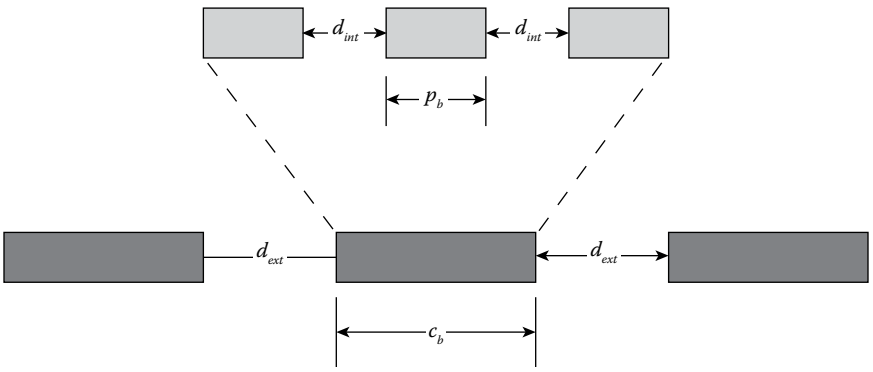

Figure 3.7 Beatha flow control basics.

uses another delay known as *external delay* (d_{ext}). Both d_{int} and d_{ext} are calculated based on the formulae explained later in this section.

Derivation of the Internal Delay (d_{int}): The reason for having internal delay is that it allows the p_b that has already been submitted to the lower layer enough time to get transferred to the other end. If T is the required time to transfer p_b from one end to another, it can be defined using the following formula:

$$T = TotalTrip \times TotalHop \times SingleHopTravelTime \tag{3.10}$$

In this equation, *TotalTrip* indicates the number of time Beatha needs to use the path from one end to another to transfer a p_b. *TotalHop* means the number of hops present on the path. Later it is denoted by *hop*. *SingleHopTravelTime* indicates the time a segment takes to travel over a single hop. It is later called *t*.

Before we move on, let us have a closer look at the capacity of a path. We know that a path having *n* hops cannot transfer *n* segments at a time [15]. Hence, the capacity of a path to simultaneously transfer segments is lower than the total number of free hops existing on that path. Figure 3.8 shows 10 different paths having different numbers of hops. It also shows the accessible hop on each path while the first hop is being used by a segment. A careful observation of this figure gives the idea that the capacity of a path is actually a series of 3. With an increment of every three hops, one extra available segment is created. Paths having one to three hops have the capacity to transfer one segment at a time, whereas paths with four to six hops have the capacity for two, those with seven to nine hops have three, ten to twelve hops have four, and so on. Therefore, the capacity of a path can be defined as follows:

$$\varsigma = \frac{hop}{3} \tag{3.11}$$

In this equation, the capacity is denoted as ζ and *hop* is the number of hops present on the path.

It was mentioned earlier that *TotalTrip* implies the amount of time Beatha needs to use a path from one end to another to transfer a p_b. The capacity of a path, however, varies (as described in Equation 3.11) and that causes *TotalTrip* to vary depending on the number of hops between two end points. Therefore, a path with ζ capacity requires the following number of trips to send a p_b:

$$TotalTrip = \frac{p_b}{\zeta} \tag{3.12}$$

The single-hop travel time (*t*) is a variable that Beatha updates on a regular basis. Upon receiving a NAK request, sender Beatha extracts the time stamp from the header to calculate the travel time (*tt*) for that particular NAK. Then it further

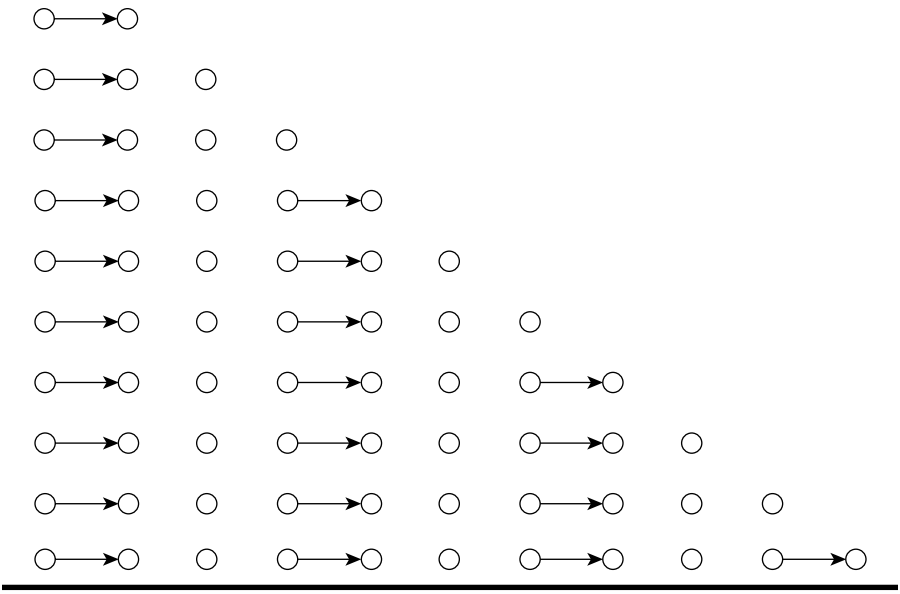

Figure 3.8 Number of segments travelling together on a path.

retrieves the number of hops that the NAK travelled on its way from the receiver's end to the sender's end. However, Beatha acquires this information from Uisce through a cross-layer information exchange. If receiver Beatha does not need to send a NAK for more than 5 seconds, it deliberately sends a message to update t at the sender's end.

$$t = 2 \times \frac{tt}{hop} \tag{3.13}$$

Replacing Equation 3.6 with Equations 3.7, 3.8, and 3.9, we get

$$T = \frac{p_b}{\varsigma} \times hop \times t \tag{3.14}$$

However, from the definition of d_{int} we know it is the time required to send a p_b. Therefore, T is actually d_{int}. So, finally we get

$$d_{int} = \frac{p_b}{\varsigma} \times hop \times t \tag{3.15}$$

Derivation of the External Delay (d_{ext}): The main motivation behind the design approach of d_{int} was to estimate the delay over an uninterrupted path. However, in reality, nodes also act as intermediate routers. It may create further delay depending

on the traffic on the path. To make the situation worse, in the event of link failure or temporary isolation, Beatha may need to slow down. Hence, its flow control process requires a feedback-based delay to control the rate adaptively and external delay (d_{ext}) serves that purpose.

The variable d_{ext} is designed in such a way that if a problem occurs and continues to give data transmission trouble, Beatha will be able to slow down its rate and gradually move back to the original flow once circumstances change. This delay incorporates three variables—*hop*, *t*, and *feedback*—and is defined as follows:

$$d_{ext} = \left(\left(hop - 1\right) \times t\right) + feedback \tag{3.16}$$

In Equation 3.16, *feedback* plays the role of adjusting d_{ext} with a change in the network topology. If Beatha suffers from segment loss, *feedback* increases itself to make d_{ext} larger, but as soon as the situation improves *feedback* starts decreasing itself to reduce d_{ext}. Because of its self-organizing nature, calculation of this variable is somewhat complicated. It works with the help of NAK and is controlled by two conditions: (i) it increases by 10% upon receipt of each NAK and (ii) it gets decreased by 50% every second. Let us consider that on an arbitrary second Beatha receives the first NAK. Thus, the change to *feedback* is as follows:

$$feedback_{new} = feedback_{old} + \left(feedback_{old} \times 0.10\right) \tag{3.17}$$

$$feedback_{new} = feedback_{old} \times \left(1 + 0.10\right) \tag{3.18}$$

If Beatha receives another NAK within that second, *feedback* looks like the following:

$$feedback_{new} = feedback_{old} \times \left(1 + 0.10\right) + \left(feedback_{old} \times \left(1 + 0.10\right) \times 0.10\right) \tag{3.19}$$

$$feedback_{new} = feedback_{old} \times \left(1 + 0.10\right) \times \left(1 + 0.10\right) \tag{3.20}$$

$$feedback_{new} = feedback_{old} \times \left(1 + 0.10\right)^2 \tag{3.21}$$

Therefore, the increment of *feedback* over a period of one second can be expressed as follows (where *n* is the number of NAKs received during that period):

$$feedback_{new} = feedback_{old} \times \left(1 + 0.10\right)^n \tag{3.22}$$

Nonetheless, according to the second condition, it gets decreased by 50% per second. Thus, this decrement can be expressed as follows:

$$feedback_{new} = feedback_{old} \times 0.5 \tag{3.23}$$

So, over a period of one second, the change in *feedback* can be expressed as follows:

$$feedback_{new} = feedback_{old} \times (1+0.10)^n - feedback_{old} \times 0.5 \tag{3.24}$$

$$feedback_{new} = feedback_{old} \times \left((1+0.10)^n - 0.5\right) \tag{3.25}$$

There might be a question as to why the increment per NAK received is 10% whereas the decrement per second is 50%. The answer lies in Equation 3.25. From this equation, it is clear that whether the value of *feedback* increases or decreases depends on n, the number of NAKs Beatha receives over a period of one second. During the development of Beatha, experiments showed that if segment loss occurs at a rate of up to four segments per second, Beatha can still operate without reducing the rate to achieve maximum throughput. However, if Beatha continues to do so for more than a four-segment loss, it bursts the network with excessive data. Based on this observation, the increment and decrement rates of feedback are set. In Equation 3.25, the multiplying factor with $feedback_{old}$—that is, $(1 + 0.10)^n - 0.5$—will remain smaller than 1 every cycle of a second for the value of n up to 4. It is always a new smaller value for $feedback_{new}$ that prevents d_{ext} from increasing. However, once n reaches 5 or more, the multiplying factor of $feedback_{old}$ grows to more than 1 and results in an increment in $feedback_{new}$. It eventually increases the value of d_{ext}. However, as soon as Beatha gets over with loss, *feedback* is exponentially decreased to trim d_{ext}. Because *feedback* is a delay, it is measured in seconds. Like all adaptive variables, *feedback* needs to be initialized before starting its operation. Its initial value is set to 0.1 second. It also maintains a lower threshold of 0.05 second—Beatha will not reduce *feedback* further once it reaches this threshold. The reason for having this lower bound is that if *feedback* ever reaches a value of 0, it will continue to be zero forever.

3.7 Experimental Setup

The proposed protocol was implemented and evaluated using the network simulator OPNET 14.0 [20]. This section describes the protocols, their configurations, scenarios, and the mobility patterns of Beatha.

3.7.1 Configurations

This simulator also provided the protocols used in the experiments except Beatha and Uisce. The implementation of Uisce was taken from an ongoing research project in the Distributed Systems Group of the Department of Computer Science at Trinity College Dublin, whereas Beatha was designed and developed by the authors. Microsoft Visual Studio.NET 2005 was used to compile the C++ implementation of all protocols used in this evaluation. The experimental hardware involved an Intel dual core 2.4 GHz machine with 3 GB RAM.

The OPNET simulator was configured with a 10×10 km^2 area and IEEE 802.11b was used as the MAC protocol. The following table shows the configuration of key parameters that were used in the experiments. The rest of the parameters were left on their default settings.

The goal of this evaluation is to compare the performance of Beatha to that of a conventional transport protocol. TCP Reno was carefully chosen as a representative of the conventional counterpart. A total of four pairs of protocols were used to conduct the experiments. Each pair was a combination of a network and a transport protocol. Beatha was paired with Uisce to form the Beatha–Uisce group, whereas TCP was paired with Dynamic Source Routing (DSR), Ad hoc On-Demand Distance Vector (AODV), and Optimized Link State Routing (OSLR) to form the TCP–DSR, TCP–AODV, and TCP–OLSR groups, respectively.

3.7.2 Simulation Scenarios

Three different scenarios were used in the experiments. The first scenario was called the *static line scenario*. This scenario was comprised of a number of paths with static nodes on them. Two end points on each path acted as the source and the destination. The number of hops on a path was varied in the experiments. Figure 3.9 shows the scenario where the nodes were placed with a gap of 70 m between each other. The transmission range of each node was set to 100 m, which is the most common range used in MANET simulations [21].

The second scenario was called the *moving line scenario*. It was comprised of eleven nodes placed as in the previous scenario; Node 1 acted as the source and Node 11 as the destination. Once the simulation began, each node started to move with a given speed ranging from 1 to 30 ms^{-1} in the same direction. When a node reached the edge of the simulation area, it hit the fence and changed its path depending on the deflecting angle.

The third and the last scenario was called the *mobility scenario*. It is illustrated in Figure 3.10, which shows how the initial placement was comprised of a 10-hop path surrounded by random nodes. On the path, two end nodes acted as the source and destination. Once the simulation began, the number of hops between the source and the destination started to change depending on the movement of the nodes.

Figure 3.9 Static line scenario.

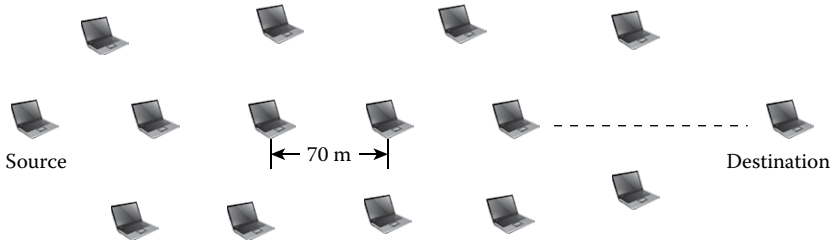

Figure 3.10 Mobility scenario.

Data rate (bps)	11 Mbps
Packet reception-power threshold (dBm)	–95
Short retry limit	7
Long retry limit	4
Max receive lifetime (secs)	0.5
Buffer size	256000
Transmit power (mW)	60

Figure 3.11 Parameter setting for IEEE 802.11b.

This scenario had two different sets of mobility patterns: the simple case, where nodes took a 90-degree turn after travelling every 100 m at a low speed between 5 and 15 ms^{-1}, and the complex case, where nodes took a 45-degree turn after travelling every 30 m at a speed of 20–30 ms^{-1}. In addition to these nodes in both cases, a total of 50 other nodes were placed in the simulation area that varied their speed between 5 and 15 ms^{-1} and changed direction randomly. Figure 3.11 shows an abstract view of this scenario.

3.8 Performance Evaluation

The objective of this performance evaluation was to analyze the Beatha protocol. Because this protocol was designed to address the problems transport protocols encounter when operating over a wireless network and quickly deploying emergency

communication, this evaluation tried to demonstrate improvement by Beatha on particular matrices that were previously hindered by those problems.

3.8.1 Evaluation of Throughput

This part of the evaluation involved two experiments. The first experiment demonstrated measurement of throughput over long multihop paths when nodes are static. It showed the effect of hop increment on throughput. The second experiment took mobility into account and demonstrated how end-to-end throughput over long multihop paths changes in the presence of mobility.

The first experiment was conducted with the static line scenario. It involved a total of 10 paths with static nodes on them. The first path had only two nodes, separating the source and the destination by a single hop. The second path had three nodes, the third had four nodes, and the tenth had eleven nodes, separating the source and the destination by 10 hops. A file of 5 MB was transferred from the source to the destination to record throughput per second. It was obtained dividing the total size of the file by the time required to transfer it. This experiment was repeated 20 times and the average was taken to plot the graphs with positive and the negative errors.

It was previously mentioned that TCP exhibits poor performance over long multihop paths for a number of reasons, including sequential medium access of the wireless MAC protocol and window-based flow control. Figure 3.12 is evidence of this claim, showing a comparison of the performance of Beatha–Uisce to that of TCP–AODV, TCP–DSR, and TCP–OLSR over multihop paths. It is clear from the graph that all three combinations of TCP achieved decent throughput that started near 400 KB/s over the single-hop path. However, their performance worsened as soon as they started operating over long multihop paths and touched a throughput level near 20 KB/s on the 10-hop path. It is also notable that among the TCP groups, the TCP–AODV pair achieved the highest throughput (23.55 KB/s).

In order to improve transport layer performance over long multihop paths, a number of changes have been introduced in Beatha, as described earlier in this chapter. Its rate-based adaptive flow control is capable of adjusting rate with the change of topology, whereas its NAK-based acknowledgment reduces reverse flow significantly. As a result, Beatha performs better than TCP while operating on the same path. This experiment showed that Beatha achieved a throughput of 819.44 KB/s over the single-hop path. Later its performance degraded, but it still maintained a data rate of 151.51 KB/s on the 10-hop path, seven times higher than the average performance of the TCP groups.

The second experiment involved measurement of throughput in the presence of mobility. The goal of this experiment was to analyze the performance of Beatha over long paths with a verity of speed in the moving line scenario. Six speed patterns were used in this experiment: 5 ms^{-1}, 10 ms^{-1}, 15 ms^{-1}, 20 ms^{-1}, 25 ms^{-1}, and 30 ms^{-1}. For each instance, 20 readings were taken and the average was used to plot the graphs.

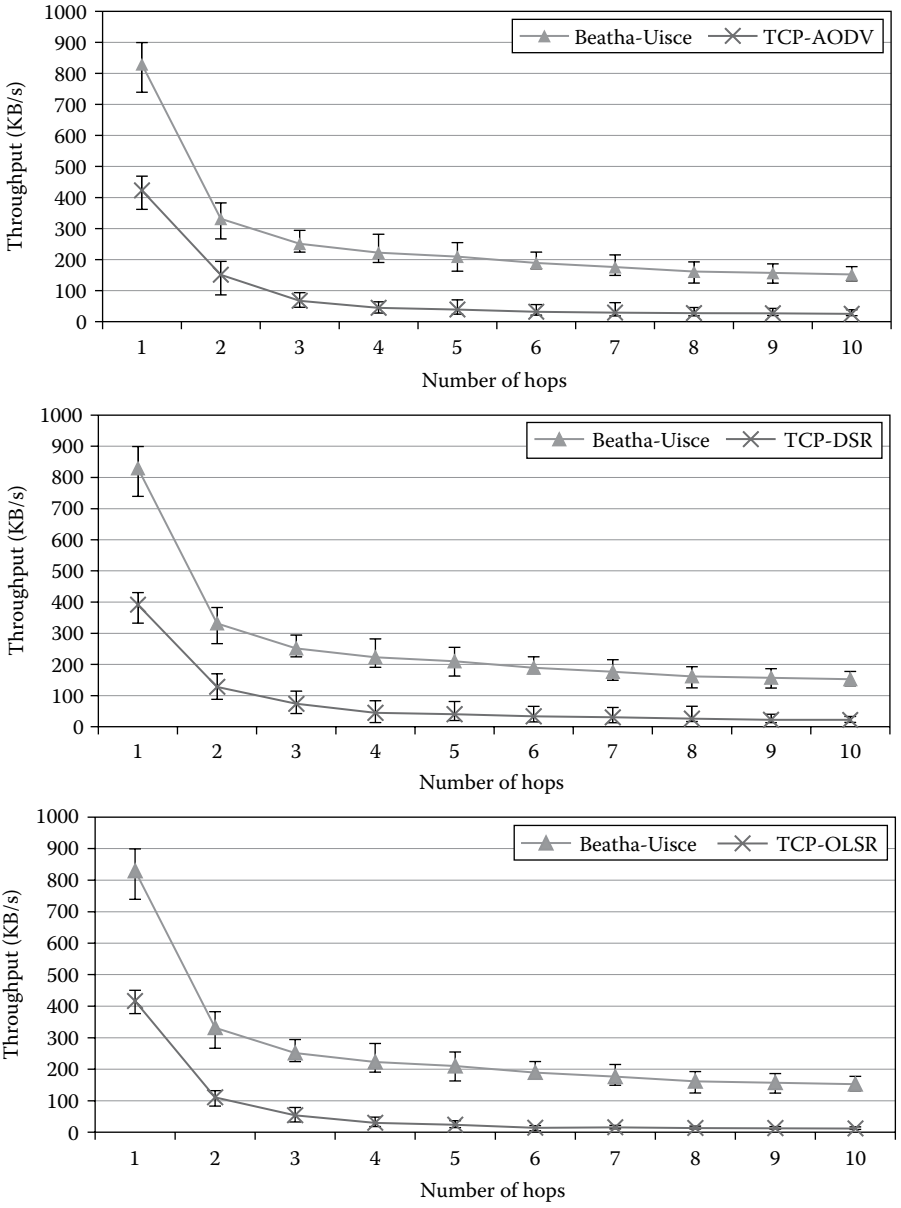

Figure 3.12 Throughput evaluation over multihop paths.

Transport protocols usually experience a drop in throughput when the nodes move with high mobility. In this kind of situation, TCP behaves differently depending on the routing protocols it uses. Figure 3.13 shows that at a lower speed the TCP–AODV pair performed better than the TCP–DSR and TCP–OLSR pairs. This is because OLSR is a proactive routing protocol that finds a path based on previously obtained information. As a result, in the event of path failure it cannot find a new path quickly. DSR is a proactive protocol, but it also uses some previously saved route caches to find paths between two end nodes. By contrast, AODV is a reactive and on-demand routing protocol that quickly recovers path failure. Because TCP performance over wireless networks is closely related to path failure [19], AODV served better than the other two pairs. Nevertheless, the performance of the TCP–AODV pair degraded at a higher speed (after 10 m/s) compared to that of the TCP–DSR pair. This is because at a higher speed, in the event of path failure, the DSR route cache frequently serves better than finding a new path.

However, this experiment clearly demonstrated that the performance of Beatha is not affected by these routing issues. Its performance is seven times better than the best TCP combination while operating with different degrees of mobility. It achieved a throughput level of 151.51 KB/s with no mobility and gradually became stable at 116.27 KB/s while operating at 30 ms^{-1} speed.

3.8.2 Evaluation of Retransmission

Two experiments were used to evaluate the retransmission performance of Beatha. These were conducted in conjunction with the previous experiments. While measuring throughput, measurements of retransmission were also made.

The first experiment was conducted in the static line scenario and the settings were identical to its experimental counterpart described earlier. It showed that TCP paired with OLSR provides the worst performance and that the TCP–DSR and TCP–AODV pairs act in a relatively similar way. However, the Beatha–Uisce pair retransmitted the least number of segments, boosting its performance to achieve the height throughput demonstrated in Figure 3.14.

The second experiment as conducted in conjunction with the second experiment described in the section "Evaluation of Throughput." The moving line scenario was used to evaluate the Beatha and TCP protocols. The settings of this experiment were the same as those described for the earlier experiment. This experiment demonstrated that the TCP–OLSR pair retransmitted more segments than any other pair but its performance was stable and did not fluctuate over the change of speed. The TCP–AODV pair suffered from more retransmissions when it operated at a speed of 10 ms^{-1} or more. However, Figure 3.15 demonstrates that Beatha performs better than any TCP pair with the least number of retransmissions while operating at different degrees of speed. Most importantly, this performance was stable regardless of the change of speed.

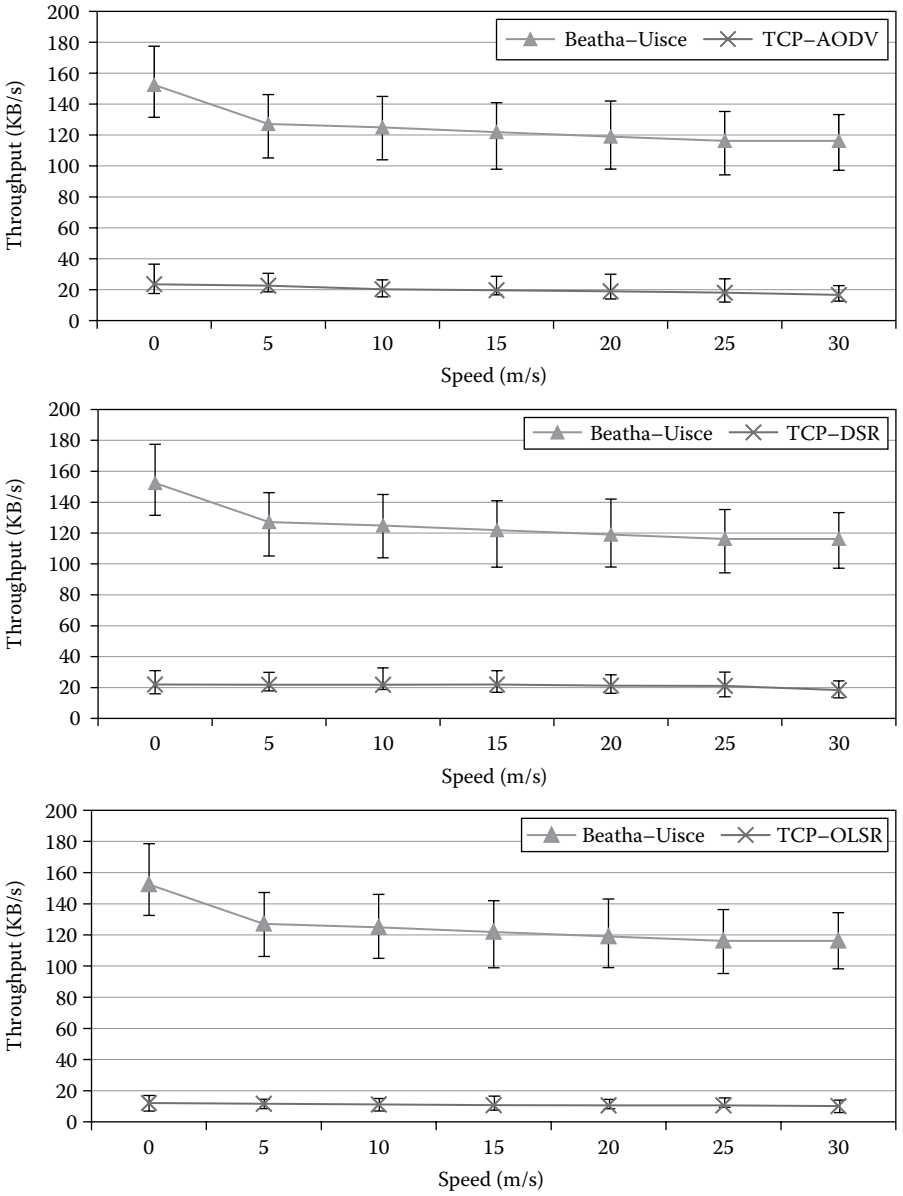

Figure 3.13 Throughput evaluation in the presence of mobility.

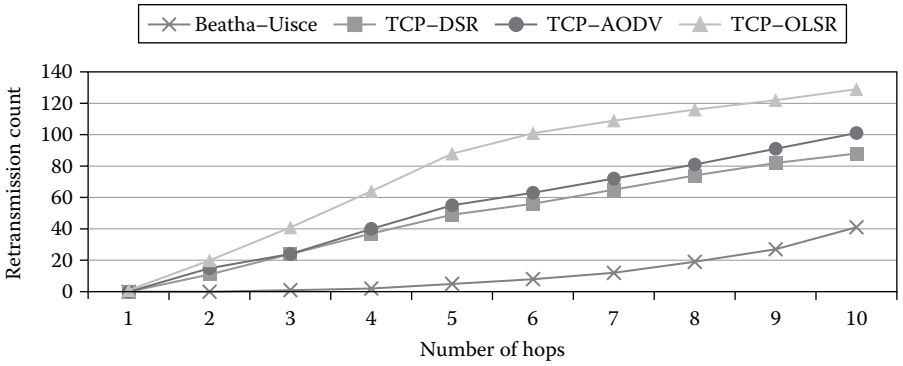

Figure 3.14 Retransmission of Beatha–Uisce and Transmission Control Protocol (TCP) with AODV, DSR, and OLSR over multihop paths.

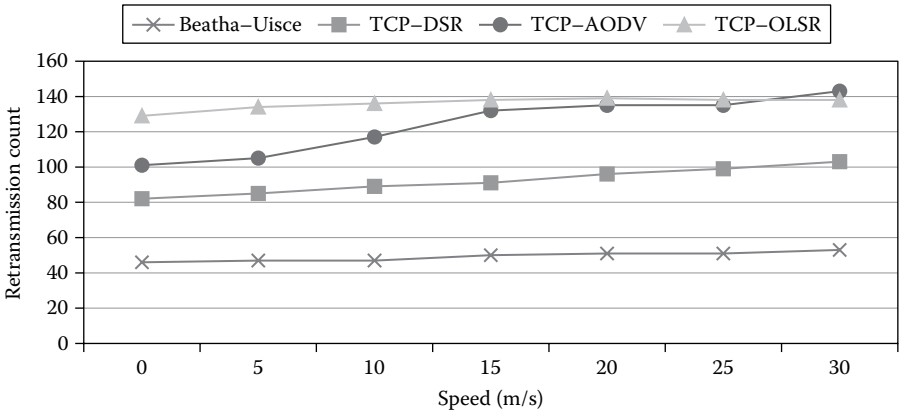

Figure 3.15 Retransmissions of Beatha and TCP in the presence of mobility.

3.8.3 Evaluation of Data Rate

This evaluation demonstrates the amount of data Beatha and TCP senders can deliver to the receiver per second. This evaluation is particularly important for demonstrating how TCP fluctuates over time while operating on long multihop paths. It also demonstrates the behavior of Beatha controlled by the flow control mechanism described earlier.

This evaluation took place in the mobility scenario. As described earlier, this scenario had two cases, one simple and one complex. There were separate experiments for each case. In order to demonstrate the exact picture, this time no average was taken; rather a first 30-second snapshot from both Beatha and TCP communication was taken to prepare the graph. The reason being,

if the average were taken then the actual fluctuation in the data rate could not be shown. From the experiments demonstrated in the sections "Evaluation of Throughput" and "Evaluation of Retransmission," it is clear that the TCP–AODV pair performs better than the other two. Therefore, only this pair was used in this evaluation.

In the first experiment, the Beatha–Uisce pair was evaluated with the TCP–AODV pair. Figure 3.16 shows the amount of data both Beatha and TCP sent per second over a period of 30 seconds. It shows that TCP touched the ground a number of times, which means that on those occasions it completely failed to send data to the receiver. Moreover, TCP could not even reach half of the minimum level Beatha maintained while operating in the same environment. In contrast, Beatha showed stable behavior, always keeping the data rate between 120 and 160 KB/s.

The second experiment was conducted in order to demonstrate the deviation of performance in both Beatha and TCP when nodes change directions frequently. Figure 3.17 shows the amount of data Beatha and TCP sent in the complex case. It shows that TCP could hardly send data and touched the ground on several occasions. In this case the nodes moved randomly, causing frequent path failure. The results show that in the same situation Beatha maintained a relatively higher margin than TCP, though it also suffered from fluctuations. Beatha's performance can be interpreted with the help of Figures 3.18 and 3.19. Because Beatha flow control is engineered by its two delays, when those delays were increased,

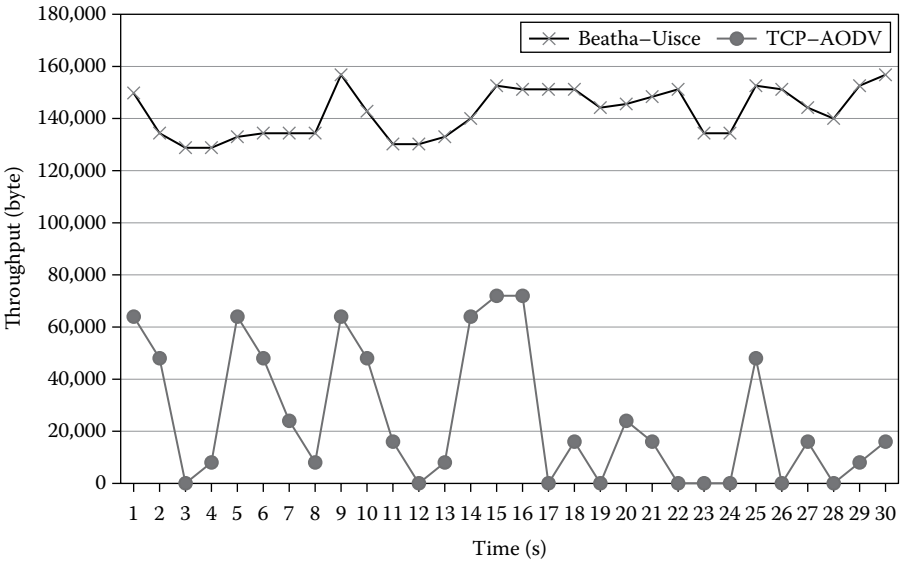

Figure 3.16 Data rate of Beatha–Uisce and TCP–AODV in the simple case.

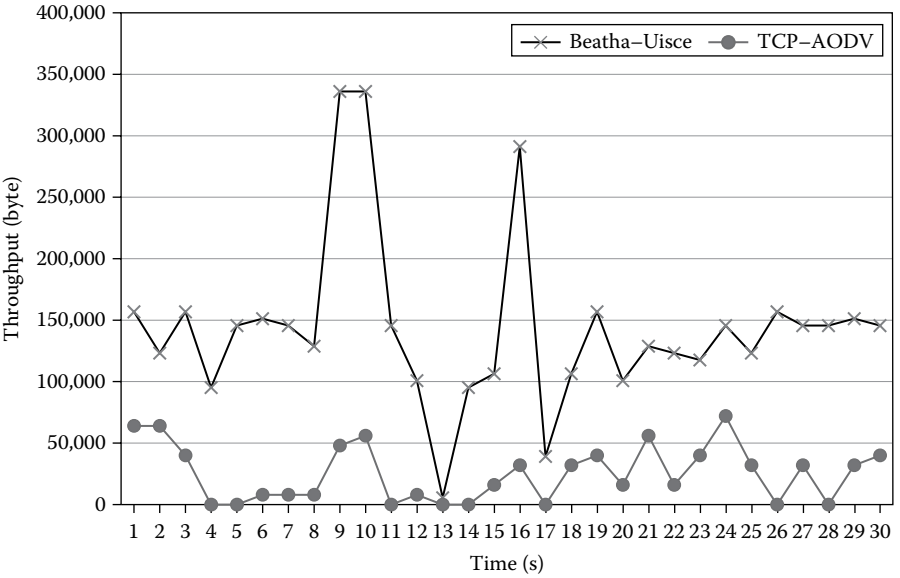

Figure 3.17 Data rate of Beatha–Uisce and TCP–AODV in the complex case.

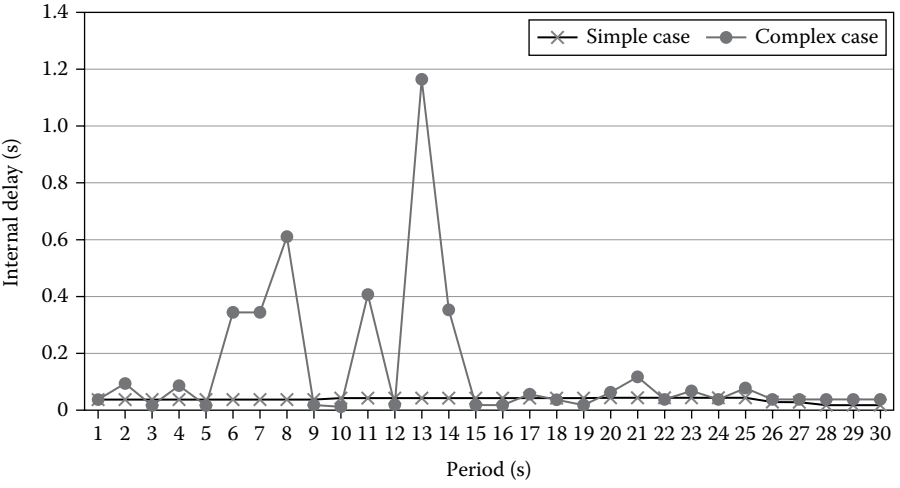

Figure 3.18 Internal delay of Beatha in the simple and complex cases.

the data rate decreased. In Figure 3.18, the internal delay reached 1.2 seconds at the 13th second of the simulation. During that period Beatha could not send any data and the data rate touched the ground (demonstrated in Figure 3.17). However, Beatha recovered from this situation very quickly and got back to a higher rate very soon because of its adaptively probing nature.

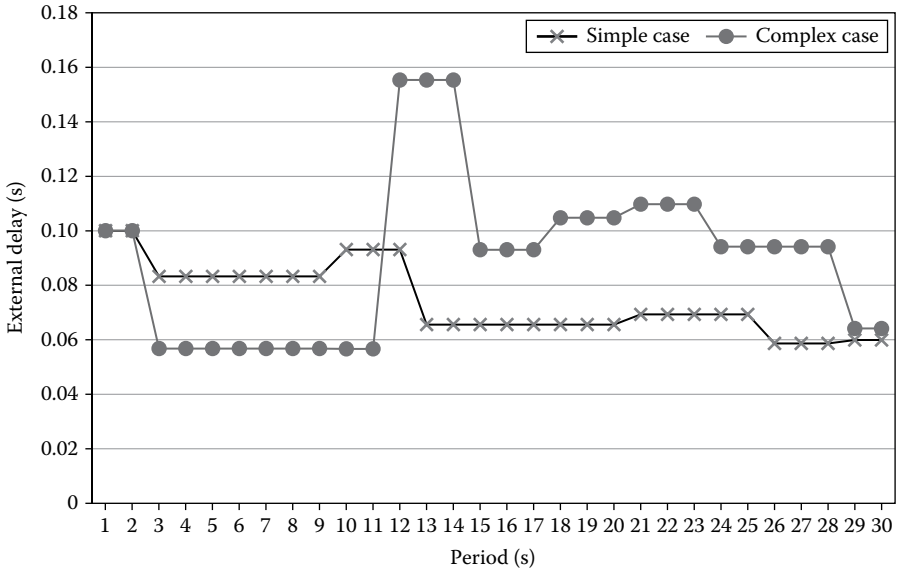

Figure 3.19 External delay of Beatha in the simple and complex cases.

3.8.4 Evaluation of Delay

This evaluation demonstrated the delay involved in delivering segments from the source to the destination node. Experiments for this evaluation took place in the mobility scenario and considered both the simple and complex cases to demonstrate the results. These experiments were conducted with the same configuration described in the section "Evaluation of Data Rate." The delay for all segments over a period of one second was accumulated first and then the average was taken by dividing it by the number of segments received during that period. As in the previous evaluation, the first 30 seconds from both the Beatha and TCP communications were taken to prepare the graphs. Figure 3.20 demonstrates that TCP exhibited mixed behavior with some fluctuations in the simple case. The graph representing TCP's performance is a broken graph, as at some points TCP failed to send any data whatsoever. However, Beatha demonstrated stable and continuous behavior without any breaks in the data transfer. Figure 3.21 demonstrates that in the complex case where nodes change their direction frequently, both Beatha and TCP exhibited poor performance. At some stage, the segment travel time in Beatha reached 1.8 seconds. Despite the fact that Beatha was having this fluctuation, its performance compared to TCP was better, as TCP failed to send data on a number of occasions. One key point in this graph is that the delay in TCP never reached especially high, but in Beatha it touched a couple of higher points. The reason for this behavior is that Beatha tries not to stop sending data even if the environment is hostile.

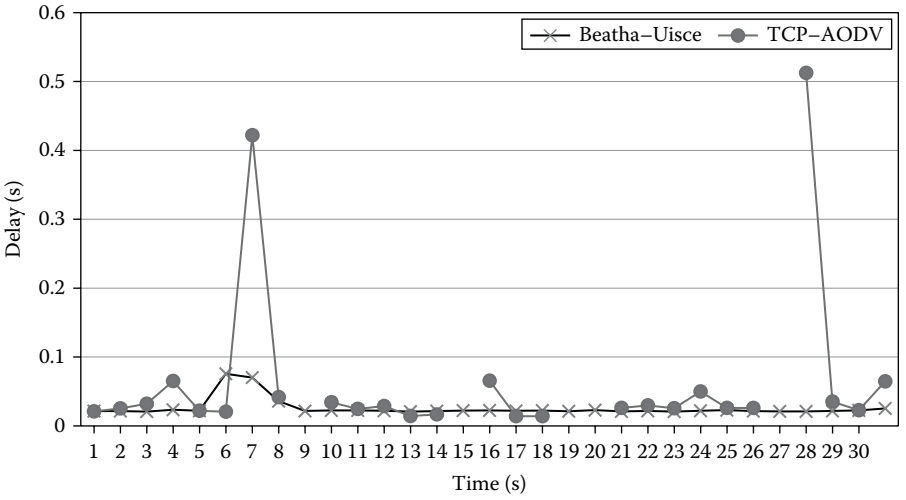

Figure 3.20 Beatha and TCP segment delay in the simple case.

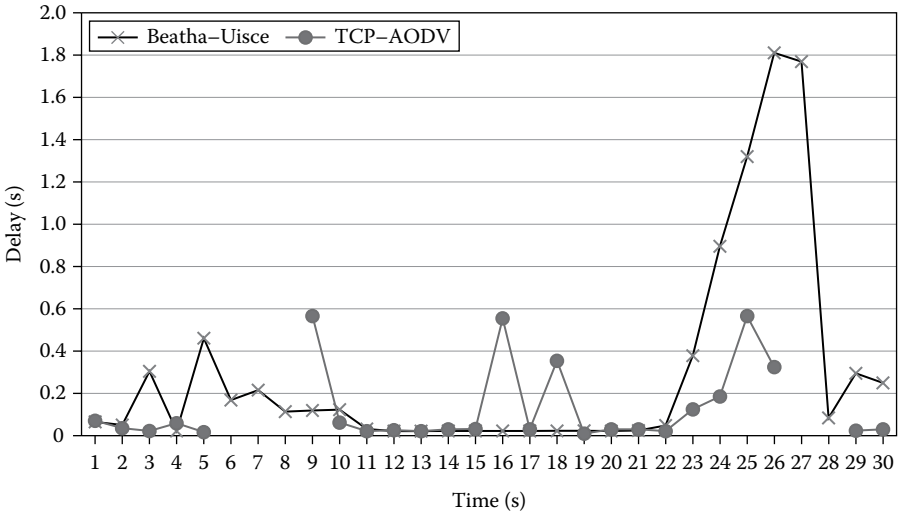

Figure 3.21 Beatha and TCP segment delay in the complex case.

3.8.5 Evaluation of Acknowledgment

This final evaluation demonstrates a comparison between Beatha and TCP based on the number of acknowledgments these two protocols sent over a particular communication. It was conducted in the static line scenario and used the same configuration as the first experiment described in the section "Evaluation of Throughput."

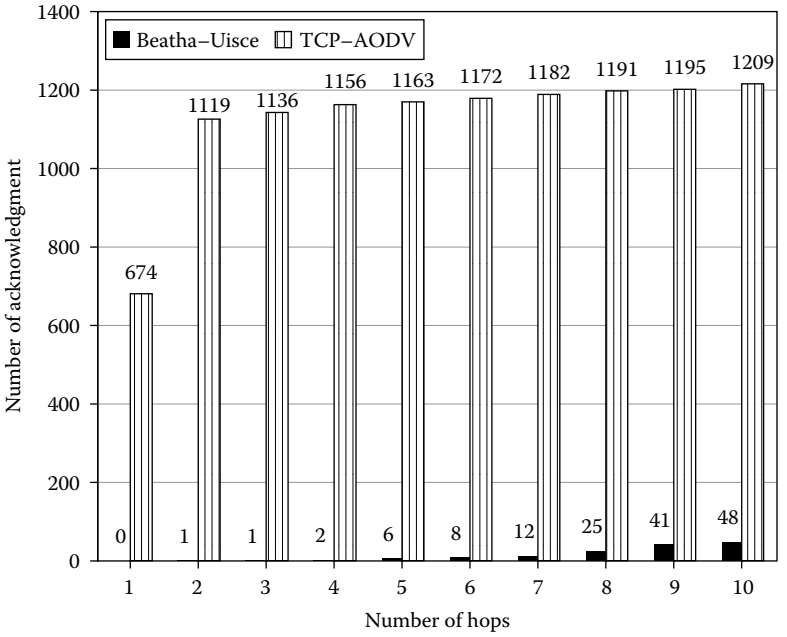

Figure 3.22 Acknowledgment sent by the receivers.

First a 5 MB file was transferred from the source to the destination, and later the number of acknowledgments required for maintaining reliability was recorded for both Beatha and TCP.

Figure 3.22 demonstrates that Beatha sent a very small number of acknowledgments for transferring a 5 MB file compared to TCP. On a single-hop path when TCP sent an average of 674 acknowledgments, Beatha sent nothing. In the worst case, on a 10-hop path Beatha sent an average of 48 acknowledgments, whereas TCP sent an average of 1209 acknowledgments.

3.9 Conclusion and Future Studies

The motivation for the work presented in this chapter arose from the observation that state-of-the-art research in supporting end-to-end reliable communication in quickly deployable wireless networks has failed to address the challenges of dynamic wireless environments where flow control needs to be adaptive and acknowledgment selective. As described in the section "Challenges," TCP exhibits poor performance in this kind of network because of its flow-control mechanism and behavior of triggering inappropriate congestion control in events where delay occurs for noncongressional reasons, mainly associated with the wireless environments. The proposed protocol addresses these problems and provides solutions to outperform

those difficulties. The contribution of this chapter is twofold: it proposes a rate-based adaptive flow-control mechanism along with a NAK-based retransmission scheme that enhances end-to-end throughput in MANETs over long multihop paths and also co-founds a non-identifier-oriented architecture, an alternative to the TCP/IP architecture, for rapidly deployable emergency communication.

There are, however, some limitations in Beatha. Its flow control and acknowledgment mechanisms are designed to assume that clocks are synchronized. However, in reality that may not be the case and therefore a clock synchronization phase is required during connection establishment. Moreover, fairness and congestion control are two areas that remain completely unaddressed. Future study of Beatha also needs to pay attention to these gaps.

References

1. S. Kurkowski, T. Camp and W. Navidi, Two Standards for Rigorous MANET Routing Protocol Evaluation, in *Proceedings of IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, Vancouver, BC, Canada, 2006.
2. N. Wang, Y. Huang and W. Liu, A Fuzzy-Based Transport Protocol for Mobile Ad Hoc Networks, in *Proceedings of IEEE International Conference on Sensor Networks, Ubiquitous and Trustworthy Computing*, Taichung, Taiwan, 2008.
3. T. Ramrekha and C. Politis, A Hybrid Adaptive Routing Protocol for Extreme Emergency Ad Hoc Communication, in *Proceedings of 19th International Conference on Computer Communications and Networks*, Zurich, Switzerland, 2010.
4. C. Fathy, T. El-Hadidi and M. El-Nasr, Fuzzy-Based Adaptive Cross Layer Routing Protocol for Mobile Ad Hoc Networks, in *Proceedings of IEEE 30th International Performance Computing and Communications Conference*, Orlando, FL, 2011.
5. A. Jangra, N. Goel and Priyanka, Efficient Power Saving Adaptive Routing Protocol (EPSAR) for Manets Using AODV and DSDV: Simulation And Feasibility Analysis, in *Proceedings of 2nd International Symposium on Intelligence Information Processing and Trusted Computing*, Hubei, China, 2011.
6. C. Murthy and B. Manoj, *Ad Hoc Wireless Networks: Architectures and Protocols*, Prentice Hall, Upper Saddle River, NJ, 2004.
7. B. Dilmaghani and R. Rao, Designing Communication Networks for Emergency Situations, in *Proceedings of IEEE International Symposium on Technology and Society*, New York, 2006.
8. M. Petrovic, V. Muthusamy and H. Jacobsen, Content-Based Routing in Mobile Ad Hoc Networks, in *Proceedings of 2nd Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, San Diego, CA, 2005.
9. K. Wang, Adaptive Attribute-Based Routing in Clustered Wireless Sensor Networks, PhD thesis, Boston University, Boston, MA, 2006.
10. C. Rezende, B. Rocha and A. Loureiro, Publish-Subscribe Architecture for Mobile Ad Hoc Networks, in *Proceedings of ACM Symposium on Applied Computing*, Ceara, Brazil, 2008.
11. J. Saltzer, D. Reed and D. Clark, End-to-End Arguments in System Design, *ACM Transactions in Computer Systems*, 2(4), pp. 277–288, 1984.

12. M. Blumenthal and D. Clark, Rethinking the Design of the Internet: The End-to-End Arguments vs. Brave New World, *ACM Transactions on Internet Technology*, 1(1), pp. 70–109, 2001.
13. A. Carzaniga, M. Rutherford and A. Wolf, A Routing Scheme for Content-Based Networking, in *Proceedings of IEEE Conference on Computer Communications* (*INFOCOM*), Hong Kong, 2004.
14. Y. Guoxian and S. Weber, Uisce: Characteristic-based Routing in Mobile Ad Hoc Networks, in *Proceedings of 10th IFIP Annual Mediterranean Ad Hoc Networking Workshop* (*Med-Hoc-Net*), Sicily, Italy, 2011.
15. G. Holland and N. Vaidya, Analysis of TCP Performance over Mobile Ad Hoc Networks. Wireless Networks, in *Proceedings of 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, 2002.
16. M. Allman, V. Paxson and W. Stevens, *RFC 2581: TCP Congestion Control*, NASA Glenn and Sterling Software, CA, 1999.
17. V. Paxson and M. Allman, *RFC 2988: Computing TCP's Retransmission Timer*, ACIRI and NASA GRC/BBN, CA, 2000.
18. J. Monks and P. Sinha, Limitation of TCP-ELFN for Ad Hoc Networks, in *Proceedings of the International Workshop on Mobile Multimedia Communication* (*MoMuC*), Tokyo, Japan, 2000.
19. G. Holland and N. Vaidya, Impact of Routing and Link Layers on TCP Performance in Mobile Ad Hoc Networks, in *Proceedings of IEEE Wireless Communications and Networking Conference* (*WCNC*), New Orleans, LA, 1999.
20. OPNET, *Modeling Concepts Reference Manual*, Technical report, OPNET Technologies, Inc., Bethesda, MD, 2008.
21. S. Kurkowski, T. Camp and M. Colagrosso, MANET Simulation Studies: The Incredibles. *Mobile Computing and Communications*, pp. 50–61, New York, 2005.