Implementing BCAST

Thomas Kunz
Carleton University

Thomas Kunz
2 Beckington Private
Ottawa, Ont., Canada
K2P 2N5

Project Manager: Louise Lamont 991-9635

Contract Number: CRC 5009870

Contract Scientific Authority: Louise Lamont 991-9635

Defence R&D Canada - Ottawa

Contract Report

DRDC-Ottawa CR 2004-xxx

March 2004

Communications Research Centre

Author

Thomas Kunz

Approved by

[Enter name here]
[Enter title here]

Approved for release by

[Enter name here]
[Enter title here]

[Enter admin info here]

[Enter standard clause here]

Abstract

Multicasting is the transmission of datagrams (packets) to a group of zero or more hosts identified by a single destination address. Maintaining group membership information and building an optimal multicast distribution structure (typically in the form of a routing tree) is challenging even in wired networks. However, nodes are increasingly mobile. One particularly challenging environment for multicast is a mobile ad-hoc network (MANET). This report discusses the implementation of a multicast protocol (`bcast`) on XORP, the eXtensible Open Router Platform from Berkeley, on Linux laptops with IEEE 802.11 interfaces. This protocol does not require the construction and maintenance of a multicast distribution structure. Rather, it broadcasts packets to all nodes in an optimized manner. Prior work, based on simulations, has shown that this approach has good overall performance. The implemented protocol supports both unreliable and reliable operation. We extensively tested our protocol in a number of ways: simulations, executing the protocol over wireless links in a static environment, and executing it in emulated mobile scenarios. In a stationary environment, all results are close to each other, and the mobility emulator introduces negligible additional overhead. In mobile scenarios, our measurements in the emulated environment differ quantitatively and qualitatively from the simulation results.

Résumé

La multidiffusion consiste en la transmission de datagrammes (paquets) à un groupe qui compte aucun hôte ou plus et qui est identifié par une seule adresse de destination. La conservation des données sur les membres d'un groupe et la construction d'une structure de multidiffusion optimale (généralement en forme d'arborescence de routage) représentent un défi, même dans les réseaux câblés. Toutefois, les nœuds sont de plus en plus mobiles. Un réseau mobile ad hoc (MANET) est un environnement particulièrement complexe pour la multidiffusion. Le présent rapport examine la mise en place d'un protocole de multidiffusion (bcast) dans XORP (eXtensible Open Router Platform), la plate-forme de routeur libre de Berkeley, sur des ordinateurs portatifs qui exécutent Linux et qui possèdent des interfaces IEEE 802.11. Ce protocole ne requiert pas la construction ni le maintien d'une structure de multidiffusion. Il diffuse plutôt les paquets à tous les nœuds d'une manière optimale. Des travaux antérieurs, fondés sur des simulations, ont démontré que cette approche produisait un rendement global satisfaisant. Le protocole mis en place supporte les opérations fiables ou non. Nous avons soumis notre protocole à des essais complexes et variés, ce qui inclut des simulations, l'application du protocole avec des liens sans fil dans un environnement statique et l'exécution du protocole dans des scénarios mobiles émulés. Dans un milieu fixe, tous les résultats se ressemblent, et l'émulateur mobile introduit une surcharge négligeable. Dans les scénarios mobiles, nos mesures de l'environnement émulé se distinguent quantitativement et qualitativement des résultats des simulations.

This page intentionally left blank.

Executive summary

Multicasting is the transmission of datagrams (packets) to a group of zero or more hosts identified by a single destination address. A multicast packet is typically delivered to all members of its destination host group with the same reliability as regular unicast packets. In the case of IP Multicasting, for example, the packet is not guaranteed to arrive at all members of the destination group or in the same order relative to other packets.

Multicasting is intended for group-oriented computing and its use within a network has many benefits. Multicasting reduces the communication costs for applications that send the same data to multiple recipients. Instead of sending via multiple unicasts, multicasting minimizes the link bandwidth consumption, sender and router processing, and delivery delay. In addition, multicasting provides a simple yet robust communication mechanism whereby a receiver's individual address is unknown or changeable transparently to the source.

There are more and more applications where one-to-many or many-to-many dissemination is an essential task. The multicast service is critical in applications characterized by the close collaboration of teams (e.g. rescue patrol, battalion, scientists, etc) with requirements for audio and video conferencing and sharing of text and images. In the Internet (IPv4), multicasting facilities were introduced via the Multicast Backbone (MBone), a virtual overlay network on top of the Internet. This overlay network consists of multicast-capable islands connected by tunnels. Each island contains one or more special routers called multicast routers, which are logically connected by these tunnels. These routers manage group membership and cooperate to route data to all hosts wishing to participate in a multicast group. IP multicast groups are identified by special IP addresses. Support for multicasting is an integral component of IPv6, so it can be assumed that multicasting applications will become even more popular with the increased popularity and acceptance of IPv6.

Typically, the membership of a host group is dynamic; that is, hosts may join and leave groups any time. There is no restriction on the location or number of members in a host group. A host may be a member of more than one group at a time. A host does not have to be a member of a group to send packets to it. A host group may be permanent or transient. A permanent group has a well-known, administratively assigned address. It is the address, not the membership of the group that is permanent; at any time a permanent group may have any number of members, even zero. Those IP multicast addresses that are not reserved for permanent groups are available for dynamic assignment to transient groups which exist only as long as they have members.

Maintaining group membership information and building an optimal multicast distribution structure (typically in the form of a routing tree) is challenging even in wired networks. However, nodes are increasingly mobile. One particularly challenging environment for multicast is a mobile ad-hoc network (MANET). A MANET consists of a dynamic collection of nodes with sometimes rapidly changing multi-hop topologies that are composed of relatively low-bandwidth wireless links. There is no assumption of an underlying fixed infrastructure. Nodes are free to move arbitrarily. Since each node has a limited transmission range, not all messages may reach all the intended hosts. To provide communication through

the whole network, a source-to-destination path could pass through several intermediate neighbour nodes. Unlike typical wireline routing protocols, ad-hoc routing protocols must address a diverse range of issues. The network topology can change randomly and rapidly, at unpredictable times. Since wireless links generally have lower capacity, congestion is typically the norm rather than the exception. The majority of nodes will rely on batteries, thus routing protocols must limit the amount of control information that is passed between nodes. Also, multicast group members and other nodes move, thus precluding the use of a fixed multicast topology.

The goal of MANETs is to extend mobility into the realm of autonomous, mobile, wireless domains, where a set of nodes form the network routing infrastructure in an ad-hoc fashion. The majority of applications for the MANET technology are in areas where rapid deployment and dynamic reconfiguration are necessary and the wireline network is not available. These include military battlefields, emergency search and rescue sites, classrooms, and conventions where participants share information dynamically using their mobile devices. These applications lend themselves well to multicast operation. In addition, within a wireless medium, it is even more crucial to reduce the transmission overhead and power consumption. Multicasting can improve the efficiency of the wireless link when sending multiple copies of messages by exploiting the inherent broadcast property of wireless transmission. While many applications, such as audio/video distribution, can tolerate loss of data content, many other applications cannot. In addition, even loss-tolerant applications will suffer a performance penalty: an audio stream may experience a short gap or lower fidelity in the presence of loss.

In previous work, we have studied a number of unicast, multicast, and broadcast routing protocols, exploring their performance in a number of multicast scenarios. Based on simulation studies, we concluded that building and maintaining multicast distribution structures is not advantageous in a MANET. Rather, a protocol that optimizes broadcasting a packet to all nodes had superior performance to all other protocols in almost all scenarios we explored. This protocol, `bcast`, is based on 2-hop neighbour information exchanged by periodic “Hello” messages.

This report discusses the implementation of this multicast protocol (`bcast`) on XORP, the eXtensible Open Router Platform from Berkeley, on Linux laptops with IEEE 802.11 interfaces. The implemented protocol supports both unreliable and reliable operation. We extensively tested our protocol in a number of ways: simulations, executing the protocol over wireless links in a static environment, and executing it in emulated mobile scenarios. In a stationary environment, all results are close to each other, and the mobility emulator introduces negligible additional overhead. In mobile scenarios, our measurements in the emulated environment differ quantitatively and qualitatively from the simulation results.

Sommaire

La multidiffusion consiste en la transmission de datagrammes (paquets) à un groupe qui compte aucun hôte ou plus et qui est identifié par une seule adresse de destination. La remise d'un paquet à diffusion sélective à tous les membres d'un groupe de multidiffusion est généralement aussi fiable que celle d'un paquet à diffusion individuelle ordinaire. Dans le cas de la multidiffusion IP, par exemple, rien ne garantit la remise du paquet à tous les membres du groupe de destination ou le maintien de son ordre initial comparativement aux autres paquets.

La multidiffusion cible l'informatique axée sur les groupes, et son utilisation dans un réseau comporte de nombreux avantages. La multidiffusion réduit les coûts de communication des applications qui envoient les mêmes données à beaucoup de destinataires. Au lieu de recourir à l'envoi répété de diffusions ponctuelles, la multidiffusion minimise la consommation de largeur de bande de liaison, le traitement de l'expéditeur et du routage ainsi que le temps de remise. En outre, la multidiffusion offre un mécanisme de communication simple mais robuste grâce auquel l'adresse personnelle du destinataire est inconnue de la source ou modifiable aisément.

Il existe de plus en plus d'applications pour lesquelles la diffusion de un à plusieurs et de plusieurs à plusieurs est une fonction essentielle. Le service de multidiffusion est essentiel aux applications caractérisées par une étroite collaboration entre les équipes (p. ex. service d'urgence, bataillon, scientifiques) et des exigences en matière de conférence vidéo et audio et de partage de textes et d'images. Dans Internet (protocole IPv4), les installations de multidiffusion ont été introduites au moyen du réseau Mbone, un réseau virtuel superposé à Internet. Ce réseau superposé est composé d'îlots fonctionnant en multidiffusion et reliés entre eux par des tunnels. Chaque îlot possède au moins un routeur spécial appelé « routeur de multidiffusion » qui est connecté de façon logique à ces tunnels. Ce routeur gère la composition des groupes et collabore afin de diriger les données à tous les hôtes qui souhaitent participer à un groupe de multidiffusion. Les groupes de multidiffusion IP sont identifiés par des adresses IP particulières. La prise en charge de la multidiffusion est une partie intégrante du protocole IPv6; il est donc permis de croire que les applications de multidiffusion seront de plus en plus utilisées avec l'adoption du protocole IPv6 et sa popularité accrue.

En général, l'adhésion à un groupe de multidiffusion est dynamique, ce qui signifie que les hôtes peuvent s'y joindre ou le quitter à tout moment. Il n'existe aucune limite concernant l'emplacement ou le nombre de membres d'un tel groupe. Un hôte peut être membre de plus d'un groupe à la fois et ne doit pas nécessairement faire partie d'un groupe pour lui envoyer des paquets. Un groupe de multidiffusion est temporaire ou permanent. Un groupe permanent possède une adresse bien connue et attribuée par l'administration. C'est l'adresse, et non l'adhésion au groupe, qui est permanente. À tout moment, un groupe permanent peut compter un nombre indéfini de membres; il peut même n'en avoir aucun. Les adresses de multidiffusion IP qui ne sont pas réservées pour des groupes permanents peuvent être attribuées de manière dynamique à des groupes temporaires qui existent aussi longtemps qu'ils ont des membres.

La conservation des données sur les membres d'un groupe et la construction d'une structure de multidiffusion optimale (généralement en forme d'arborescence de routage) représentent un défi, même dans les réseaux câblés. Toutefois, les nœuds sont de plus en plus mobiles. Un réseau mobile ad hoc (MANET) est un environnement particulièrement complexe pour la multidiffusion. Un MANET consiste en un regroupement dynamique de nœuds avec des topologies à plusieurs bonds qui changent parfois rapidement et qui sont composées de liens sans fil à débit relativement bas. Il n'y a aucune infrastructure fixe sous-jacente. Les nœuds peuvent se déplacer librement. Puisque chaque nœud possède une portée d'émission restreinte, il est possible que les destinataires visés ne reçoivent pas tous les messages. Pour transmettre les communications à l'ensemble du réseau, la voie de la source à la destination peut emprunter plusieurs nœuds intermédiaires voisins. Contrairement aux protocoles de routage câblé typiques, les protocoles de routage ad hoc doivent résoudre une vaste gamme de problèmes. La topologie du réseau peut changer rapidement et de manière aléatoire à n'importe quel moment. Puisque les liens sans fil ont généralement une capacité inférieure, la congestion est devenue la norme plutôt que l'exception. La majorité des nœuds sont alimentés par des piles; les protocoles de routage doivent donc limiter le volume d'information de contrôle transmis entre les nœuds. En outre, les membres du groupe de multidiffusion et les autres nœuds se déplacent, ce qui empêche l'utilisation d'une topologie de multidiffusion fixe.

L'objectif des MANET est d'étendre la mobilité aux domaines autonome, mobile et sans fil, où un ensemble de nœuds forment une infrastructure d'acheminement des données de manière ponctuelle. La plupart des applications destinées à la technologie des MANET sont dans des zones qui exigent une mise en place rapide et une reconfiguration dynamique et pour lesquelles aucun réseau sans fil n'est disponible. Cela inclut les champs de bataille, les sites de recherche et de sauvetage, les salles de classe et les congrès où les participants partagent des renseignements de façon dynamique à l'aide de leurs appareils mobiles. Ces applications conviennent parfaitement à la multidiffusion. En outre, dans un appareil sans fil, il est encore plus important de réduire la surcharge de transmission et la consommation d'énergie. La multidiffusion peut améliorer l'efficacité des liaisons sans fil lors de l'envoi d'un grand nombre de messages en exploitant la propriété inhérente de diffusion générale des transmissions sans fil. Bien que de nombreuses applications, comme la distribution audio et vidéo, puissent supporter une perte de contenu d'information, beaucoup d'autres applications ne peuvent tolérer cette situation. De plus, même les applications qui peuvent supporter de telles pertes verront leur rendement diminuer : le flot de données audio peut comporter des lacunes ou connaître une diminution de sa fidélité en présence de pertes.

Lors de travaux précédents, nous avons étudié un certain nombre de protocoles de routages à diffusion individuelle, à diffusion sélective et à diffusion générale. À la lumière des études sur les simulations, nous avons conclu que la construction et le maintien d'une structure de multidiffusion ne sont pas avantageux dans un MANET. Un protocole qui optimise la diffusion d'un paquet à tous les nœuds obtient un rendement supérieur à tous les autres protocoles dans la presque totalité des scénarios étudiés. Ce protocole, appelé « bcast », est fondé sur de l'information voisine à deux bonds échangée par des messages de « salutation » périodiques.

Le présent rapport examine la mise en place d'un protocole de multidiffusion (bcast) dans XORP (eXtensible Open Router Platform), la plate-forme de routeur libre de Berkeley, sur des ordinateurs portatifs qui exécutent Linux et qui possèdent des interfaces IEEE 802.11. Le

protocole mis en place supporte les opérations fiables ou non. Nous avons soumis notre protocole à des essais complexes et variés, ce qui inclut des simulations, l'application du protocole avec des liens sans fil dans un environnement statique et l'exécution du protocole dans des scénarios mobiles émulés. Dans un milieu fixe, tous les résultats se ressemblent, et l'émulateur mobile introduit une surcharge négligeable. Dans les scénarios mobiles, nos mesures de l'environnement émulé se distinguent quantitativement et qualitativement des résultats des simulations.

Table of contents

List of figures

List of tables

Acknowledgements

This work is funded by the Defence Research and Development Canada (DRDC) and benefited from various discussions with members of the CRC RNS mobile networking group.

1. Introduction

Multicasting is the transmission of datagrams (packets) to a group of zero or more hosts identified by a single destination address. A multicast packet is typically delivered to all members of its destination host group with the same reliability as regular unicast packets. In the case of IP Multicasting, for example, the packet is not guaranteed to arrive at all members of the destination group or in the same order relative to other packets.

Multicasting is intended for group-oriented computing. There are more and more applications where one-to-many or many-to-many dissemination is an essential task. The multicast service is critical in applications characterized by the close collaboration of teams (e.g. rescue patrol, battalion, scientists, etc) with requirements for audio and video conferencing and sharing of text and images. In the Internet (IPv4), multicasting facilities were introduced via the Multicast Backbone (MBone), a virtual overlay network on top of the Internet. This overlay network consists of multicast-capable islands connected by tunnels. Each island contains one or more special routers called multicast routers, which are logically connected by these tunnels. These routers manage group membership and cooperate to route data to all hosts wishing to participate in a multicast group. IP multicast groups are identified by special IP addresses. Support for multicasting is an integral component of IPv6, so it can be assumed that multicasting applications will become even more popular with the increased popularity and acceptance of IPv6. Note that the acceptance and use of group-related applications is not only based on technological criteria. [Grudin 2002] for example discusses some sociological issues relevant to the design and use of group applications.

Typically, the membership of a host group is dynamic; that is, hosts may join and leave groups any time. There is no restriction on the location or number of members in a host group. A host may be a member of more than one group at a time. A host does not have to be a member of a group to send packets to it. A host group may be permanent or transient. A permanent group has a well-known, administratively assigned address. It is the address, not the membership of the group that is permanent; at any time a permanent group may have any number of members, even zero. Those IP multicast addresses that are not reserved for permanent groups are available for dynamic assignment to transient groups which exist only as long as they have members. RFC 1700 [RFC 1700] lists well-known multicast addresses for IPv4 as of 1994. More recently, these addresses, like all other well-known (assigned) numbers are managed by an online database, accessible through a web page (`www.iana.org`). RFC 3171 [RFC 3171] documents the guidelines employed but IANA, the Internet Assigned Numbers Authority, in assigning such well-known IPv4 multicast addresses. Based on those well-known multicast addresses, RFC 2375 [RFC 2375] suggests similar well-known multicast IP addresses for IPv6. RFC 2908 [RFC 2908] proposes a general multicast address allocation architecture for the Internet, and is intended to be generic enough to apply to both IPv4 and IPv6 environments. RFC 3306 [RFC 3306] introduces encoded information in the multicast address to allow for dynamic allocation of IPv6 multicast addresses and IPv6 source-specific multicast addresses. Finally, RFC 3307 [RFC 3307] specifies guidelines for allocating permanent IPv6 multicast addresses, dynamic IPv6 multicast addresses, and permanent IPv6 multicast group identifiers. The purpose of these guidelines is to reduce the probability of IPv6 multicast address collision, not only at the IPv6 layer, but also at the link-layer of media that encode portions of the IP layer address into the link-layer address.

The use of multicasting within a network has many benefits. Multicasting reduces the communication costs for applications that send the same data to multiple recipients [Varshney 2002]. Instead of sending via multiple unicasts, multicasting minimizes the link bandwidth consumption, sender and router processing, and delivery delay. In addition, multicasting provides a simple yet robust communication mechanism whereby a receiver's individual address is unknown or changeable transparently to the source.

Maintaining group membership information and building an optimal multicast distribution structure (typically in the form of a routing tree) is challenging even in wired networks. A very detailed survey of the work done in that area and a discussion of various design trade-offs can be found in [Li 2002]. However, nodes are increasingly mobile. One particularly challenging environment for multicast is a mobile ad-hoc network (MANET). A MANET consists of a dynamic collection of nodes with sometimes rapidly changing multi-hop topologies that are composed of relatively low-bandwidth wireless links. There is no assumption of an underlying fixed infrastructure. Nodes are free to move arbitrarily. Since each node has a limited transmission range, not all messages may reach all the intended hosts. To provide communication through the whole network, a source-to-destination path could pass through several intermediate neighbour nodes. Unlike typical wireline routing protocols, ad-hoc routing protocols must address a diverse range of issues. The network topology can change randomly and rapidly, at unpredictable times. Since wireless links generally have lower capacity, congestion is typically the norm rather than the exception. The majority of nodes will rely on batteries, thus routing protocols must limit the amount of control information that is passed between nodes. Also, multicast group members and other nodes move, thus precluding the use of a fixed multicast topology. [Kunz 2002] provides an overview of some best-effort IP multicast routing protocols for fixed networks and the evolution of these protocols as hosts and finally all nodes (including intermediate routers) become mobile.

The goal of MANETs is to extend mobility into the realm of autonomous, mobile, wireless domains, where a set of nodes form the network routing infrastructure in an ad-hoc fashion. The majority of applications for the MANET technology are in areas where rapid deployment and dynamic reconfiguration are necessary and the wireline network is not available. These include military battlefields, emergency search and rescue sites, classrooms, and conventions where participants share information dynamically using their mobile devices. These applications lend themselves well to multicast operation. In addition, within a wireless medium, it is even more crucial to reduce the transmission overhead and power consumption. Multicasting can improve the efficiency of the wireless link when sending multiple copies of messages by exploiting the inherent broadcast property of wireless transmission.

RFC 3170 [RFC 3170] describes the challenges involved with designing and implementing multicast applications. The document lists a number of multicast applications and derives unique multicast service requirements for various groups of applications. While many applications, such as audio/video distribution, can tolerate loss of data content, many other applications cannot. In addition, even loss-tolerant applications will suffer a performance penalty: an audio stream may experience a short gap or lower fidelity in the presence of loss. Among the loss intolerant application categories are file distribution and caching, monitoring applications (stock prices, sensor readings, etc.), synchronized resources (directories, distributed databases, etc.), concurrent processing, collaboration/shared document editing, and online auctions. A similar discussion of multicast applications and their requirements can be found in [Varshney 2002]. Some of the loss-intolerant applications discussed in these documents are relevant in a MANET environment as well (such as the collaboration, caching/file distribution, or monitoring applications). In addition,

MANET-specific applications such as military command-and-control applications also require a high degree of reliability.

This report is organized as follows. Section 2 summarizes earlier results about multicasting in MANETs and presents `bcast`, the multicast routing protocol we choose to implement. Section 3 discusses the design of `bcast`'s implementation on top of XORP. Section 4 describes a number of tests we conducted to compare the protocol performance in a real testbed with the results predicted by a network simulator, NS2. Finally, Section 5 summarizes our work and draws conclusions for future research.

2. Multicasting in MANETs

2.1 Literature survey

Achieving reliable packet delivery in a MANET is not trivial. A few simulation studies have explored the performance of MANET multicast routing protocols such as the multicast extensions for AODV and ODMRP [Cheng 2001, Ding 2002, Zhu 2002]. These studies commonly simulated an area of usually 1000 x 1000 meters, populated by 50 mobile nodes. Nodes move according to the "random waypoint" mobility model: initially, nodes are placed randomly within the area. Each node picks a destination and moves to that destination based on a speed that is uniformly distributed between 0 and MAX. Once a node reaches the destination, it pauses for PAUSE seconds, after which the process repeats itself. In all these studies, nodes communicated over an IEEE 802.11 wireless link of 2 Mbps, the radio range was 250m. Only a subset of the MANET nodes joined a single multicast group, with some of these nodes sending fixed size data packets to all other nodes at a constant rate (i.e., CBR traffic).

The results show that the packet delivery ratio does drop below 20% (i.e., only 1 out of every 5 packets is, on average, received by a multicast receiver) for a large number of senders [Ding 2002]. In environments with one or a few senders, packet delivery ratios as low as 25% were observed when the mobility rate increased (nodes moved constantly, with MAX speed set to 20 m/s) [Cheng 2001]. Possible improvements, such as pro-actively predicting link breakage and maintaining the multicast distribution data structure before links break (and therefore packets get lost) can increase the packet delivery ratio, but under high mobility scenario, it is still below 90% [Zhu 2002]. In a nutshell, all these protocols exhibit intolerably high packet loss rates under moderate to high mobility rates. Similarly poor results are shown in [Lee 2000] for a number of other multicast routing protocols.

Building and maintaining a multicast distribution structure (typically a tree or mesh) in a MANET with its highly dynamic topology introduces its own complexities and overheads (control messages, data structures at intermediate nodes, etc.). Based on the above studies, this effort does not necessarily result in good performance; it therefore becomes questionable whether it is indeed worth the effort. Following this line of thought, some researchers have explored whether broadcasting/flooding a MANET with packets could be a viable alternative to ensure high packet delivery ratios. The results in [Obraczka 2001a, Obraczka 2001b] indeed show that flooding results in higher packet delivery ratios then ODMRP, which in turn outperforms AODV. But in the scenarios studied in these papers, flooding could result in packet delivery ratios as low as 70%, leading the authors to conclude that "*even flooding is insufficient for reliable multicast in ad hoc networks when mobility is very high*" [Obraczka 2001b, page 627]. In addition, the simulation scenarios were all based on the assumption that every node in the MANET was interested in the data packets (i.e., a global broadcast). More generally, only a subset of nodes will be interested in any specific multicast group, flooding the data to all nodes may induce a high overhead, negating one of the stated advantages of multicasting (see above).

This network overhead was exacerbated in the two studies by the fact that the packet broadcast was implemented in a trivial manner; with every node re-broadcasting a packet the first time it receives it. As discussed in [Williams 2002] and [Lou 2002], more efficient broadcast algorithms can be implemented. However, even those algorithms will suffer from low packet delivery ratios (60%-80%) as the severity of the network environment (mobility rate, traffic load, etc.) increases.

In conclusion, it seems that flooding/broadcasting data in a MANET is not sufficient to ensure high packet delivery ratios. While flooding is attractive due to its absence of a multicast distribution structure (and its ensuing maintenance), it may lead to high network traffic when propagating data packets to nodes that are not interested in it. To examine this issue further, the author conducted a detailed simulation study [Kunz 2003]. Initially, the study explored the performance of a number of best-effort protocols: 2 unicast routing protocols, 3 multicast routing protocols, and 2 broadcast protocols. The extensive simulation results show that broadcast protocols perform surprisingly well, and that this performance does not come with a high overhead.

2.2 The `bcast` protocol

Based on our initial simulation results [Kunz 2003], we focused our attention on one of the protocols that performed best across a wide range of scenarios: `bcast`. This protocol implements a scalable broadcast algorithm similar to the algorithm described in [Lou 2002]. `bcast` uses 2-hop neighbor knowledge that is exchanged by periodic "Hello" messages. Each "Hello" message contains the node's identifier (IP address) and its list of known neighbors. After a node receives a "Hello" packet from all its neighbors, it has two-hop topology information. If node B receives a broadcast from node A, B knows all neighbors of A. If B has neighbors not covered by A, it schedules the broadcast packet with a random delay. If, during the delay, B receives another copy of this broadcast from C, it can check whether its own broadcast will still reach new neighbors. If this is no longer the case, it will drop the packet. Otherwise, the process continues until B's timer goes off and B itself rebroadcasts the packet.

One issue to be solved is the determination of the random delay. The original authors of that protocol suggested a dynamic strategy. Each node searches its neighbor table for the maximum neighbor degree of any neighbor node, MAX. If its own node degree is N, it calculates the random delay as MAX/N. This is a greedy strategy: nodes with the most neighbors usually broadcast before others.

In our implementation, HELLO messages are scheduled with uniform distribution in the interval [1.5 seconds, 2.5 seconds]. A node is not considered a neighbor anymore if we miss the next HELLO message. Preliminary experiments indicated that these parameter settings result in good performance. Packet broadcast is delayed to allow for a more aggressive drop strategy. As discussed above, the idea is to allow nodes with many neighbors to broadcast first by using a scaling factor of MAX/N. To explore the effect of this factor, we scaled this factor with 100 ms and 10 ms. In the former case, packet latencies will increase, but there is better chance that packet broadcasts are cancelled, resulting in lower overheads and potentially higher packet delivery ratios.

The basic mechanism to improve the packet delivery ratio in `bcast` is fairly straightforward: every node buffers the last X packets. X can be any arbitrary number, to keep the memory requirement at each node low, we set X to a small number (30 packets) in our code. The buffer is implemented in FIFO fashion, storing the last X unique packets a node received (from the same or a different sender, in order or out of order).

When a node receives a packet with sequence number S from source node SRC, it checks whether it also received packet S-1 from the same source. If not, a node issues a 1-hop broadcast to the neighbors, asking for retransmission of this packet (the NACK message). Each neighbor, upon receiving the NACK packet, checks its local cache and retransmits the packet (assuming it

has it in its local buffer). To reduce collisions, the NACKs and the packet retransmissions are jittered randomly by 10 milliseconds. In addition, NACKs have a timeout mechanism associated with them, so even if a NACK or retransmission is lost, packets can be recovered. NACKs are re-issued up to a certain maximum number of attempts (up to 3 times in our implementation).

To reduce network traffic, nodes with pending packet retransmissions will cancel their retransmission if they overhear another node X re-broadcasting this packet. This is based on the assumption that the requesting node will receive this packet as well, satisfying the NACK. This is arguably not guaranteed to be the case: node X could be out of reach of the requesting node, broadcasting packet N for other reasons. However, with multiple NACK attempts (spaced apart multiple seconds), eventually only nodes that received a NACK will attempt to re-transmit a packet. Since they received the NACK, and packets are retransmitted with little additional delay, it is reasonable to assume that the requesting node, in turn, will receive their transmission.

If a sequence of packets is lost, our NACK mechanism recovers from this by backtracking. Assume that packets 3-6 from source S are lost. When a node receives packet 7 from S, it will trigger a NACK for packet 6. Once packet 6 is received, a NACK for packet 5 is triggered, etc. Finally, upon receiving a re-transmitted packet 3, the node determines that it already received packet 2 and the backtracking will stop. Note that this backtracking will also terminate once a packet cannot be recovered. For example, if the node is unable to recover packet 4, it will not check for receipt of packet 3, and therefore not trigger a NACK for packet 3. All else being equal, this is not unreasonable: since nodes buffer the most recent packets, being unable to retrieve packet 4 from a neighbor is a strong indication that older packets (such as packet 3) may also not be available from those neighbors.

With this NACK based scheme, the packet delivery ratio can be improved. For example, we found through simulations (reported in [Kunz 2003]) that in a 1 sender/10 receiver scenario, the packet delivery ratio increases from 99.653% to 99.822% under low mobility. In 8 out of 10 simulation scenarios, the packet delivery ratio was a perfect 100%. However, there are still sources of packet loss that a NACK-based scheme cannot completely avoid: problems due to the NACK mechanism itself and long-lived network partitions.

1. Persistent loss of NACK/retransmission

   Due to collisions, the NACK or packet re-broadcast may be lost. In this case, the mechanism fails. We improve the reliability of the NACK mechanism by using a timer and re-issuing NACKs if needed. Since, for reasons discussed below, not every packet is recoverable, we keep the number of NACK retransmission attempts to a small number (3) to avoid inducing a high load on the network.

2. Network partitions

   Due to the dynamic nature of MANETs, the network may partition for a lengthy period of time. Once the partitions merge, nodes recover, by backtracking, from a number of missed packets, but with nodes only buffering up to the 30 most recent data packets, the recovery is limited by the buffer size and the number of multicast senders. In essence, with a single multicast sender, 30 packets allow to bridge network partitions of up to 3 seconds (assuming a 10 packets/second traffic rate). To recover from a network partition of say 120 seconds, the nodes would have to buffer approximately 1200 or more packets, more if there was more than one multicast sender.

Even if nodes were to buffer many packets, this does not guarantee 100% reliable data delivery. If partition occurs towards the end of the simulation, nodes may miss out on the last packets. However, in the absence of a new packet with higher sequence number, this situation is indistinguishable from the case of the sender having stopped transmitting, so no NACKs are triggered (nor would this help, since the node remains disconnected from the network).

We conducted some simulations based on the mechanisms described above: every node buffers the 30 most recent packets, NACKs are re-issued if we fail to receive a packet for up to three times, with a NACK timeout value of 1 second. Due to the above reasons, we did not expect a perfect 100% packet delivery ratio, however, we expected an improved packet delivery rate across all scenarios. Upon closer examination of the results, we discovered that the NACK mechanism potentially results in many NACKs. This high number of NACKs (plus the packet retransmission they trigger) substantially adds to the network load, resulting in congestion and packet losses. It is clearly apparent from these numbers that NACKs have to be rate controlled: if the network is busy, aggressively asking for packet retransmissions makes a bad situation only worse.

After exploring a number of options, we settled in the following mechanism to limit the number of NACKs: each node monitors the traffic density by keeping track of when it sends a packet or receives one. This is done in a sliding window of fixed size. Before sending a NACK, the node checks how long ago the earliest packet in that sliding window was send/received. If this period is too short (indicating a high network traffic load), the NACK is suppressed. The size of the sliding window (30) and the minimum required time difference (0.7 secs) are derived at experimentally and balance the need for allowing many NACKs to go ahead with the need to prevent congestion in a heavily loaded network.

3. bcast Implementation for XORP

The XORP project [Xorp 2003] is developing an open source software router. The software is intended to be stable and fully featured enough for production use, and flexible and extensible enough for research use. As of release 0.5 from November 2003 (the version our work is based upon), XORP runs natively on FreeBSD and Linux, with support for other platforms and the Click Modular Router [Click 2003] expected in the future. Some more information about the XORP platform is provided in Appendix A.

3.1 Interaction with XORP

bcast becomes yet another user process implementing routing functionality in XORP. The bcast process gets started by rtrmgr. However, we do not build explicit routing table entries and communication between the bcast processes on different hosts is done via sockets. While the XORP design document claims that the FEA provides an abstract I/O facility for routing protocols, feedback from the XORP developers confirmed that this is less stable and feature-rich than one may wish for in the XORP version we used (0.5, expected to be replaced with XORP 1.0 sometime in the spring of 2004).

To integrate bcast into XORP, we need to define an interface, template file, and appropriate entries in the router configuration file. Communication between XORP processes is based on XORP Resource Locators or XRLs, which are generated automatically from interface descriptions.

In XORP terminology, the bcast process is an *XRL Target*: something that XRL requests can be directed to in order to be executed. Multiple XRL targets can exist within a single process, though there will typically be one target per process. Each XRL Target has an associated name and a set of IPC mechanisms that it supports. At start-up each target registers its name and IPC mechanisms with the Finder, so other processes are able to direct requests to it.

An *XRL Interface* is a set of related methods that an XRL Target would implement. Each XRL Interface is uniquely identified by a name and version number. An XRL Target will nearly always implement multiple interfaces and potentially multiple versions of particular interface. For instance, every XRL Target could implement a process information interface that would return information about the process hosting the target. Each XRL Target will implement interfaces specific to their field of operation: a routing process would implement the “routing process interface” so that the RIB process can make identity-agnostic XRL calls for tasks like redistributing a route to another routing protocol.

The interface definition for bcast (see Appendix B) specifies XRLs to set and retrieve 4 protocol parameters: the HELLO interval, the buffer size in case of the reliable version of the protocol, whether the protocol should be run in the reliable version, and the wireless interface over which the protocol should operate. The last method is used by XORP to activate the protocol instance by rtrmgr, after instantiating all XORP processes.

When rtrmgr starts, it first reads a template file for each process and then a configuration file. The template file for bcast is listed in Appendix B, specifying the configurable options for the protocol and default values for them. In addition to specifying the configurable options, the

template file also specifies what the rtrmgr should do when an option is modified in the configuration file. To instantiate XORP with bcast, the configuration file listed in Figure 1 can be used. In this file, the default parameters are overwritten with new values (instructing rtrmgr to create an instance of bcast that sends HELLO messages on average every 2 seconds, use a buffer size of 30 packets, run in the reliable version, and use the wireless interface "ath0"). The rtrmgr will create all necessary router processes, invoke the appropriate XRLs to set the configurable options, and activate the modules.

```
protocols {
  bcast {
    hello-int: 2
    buffer-size: 30
    reliability: true
    proto_intf: "ath0"
  }
}
```

Figure 1: rtrmgr Configuration File

3.2 bcast Internals

The main idea behind the protocol is to intercept multicast packets and selectively re-broadcast them. To identify duplicate and missing multicast packets, a multicast sender has to prefix each multicast packets with the bcast protocol header, in addition to the IP and UDP header, see Appendix B.

bcast is implemented as a collection of classes, each implementing a specific set of functionalities. The main class is XrlBcastTarget which implements methods to deal with all XRLs (files bcast_tgt.[cc,hh]). This code also deals with the transmission and reception of all control messages (HELLO, NACK), and the interception of multicast packets. The class UniqueIDs (in files unique_id.[cc,hh]) implements a data structure to keep track of recently seen multicast packets. Class Neighbors (in files neighbors.[cc,hh]) manages the 1-hop and 2-hop neighbor information for each node. Class bcast_buffer (in files buffer.[cc,hh]) manages the list of pending multicast packets that are waiting to be re-broadcast, and class bcast_cache (in files (cache.[cc,hh]) keeps a FIFO list of recently transmitted multicast packets to respond to NACKs.

bcast uses two primary IPC mechanism. Each node joins a multicast group, and control messages are transmitted over a socket with TTL set to 1, to limit the propagation of these messages to immediate neighbors. In addition, bcast uses the pcap [Pcap 2003] library to intercept multicast packets, cache them, and if appropriate re-broadcast them after some delay. To intercept IP multicast packets, bcast employs the following pcap filter expression:

```
#define FILTER \
  "(ip multicast and not dst net 224.0.0.0 and not dst host 255.255.255.255)"
```

This filter captures all IPv4 multicast packets that are not sent to an IP multicast address in the range 224.0.0.0 – 224.0.0.255 (which are reserved by IANA for a number of protocols) and that are not IP broadcast packets).

The protocol implementation also uses a number of compile-time constants defined in bcast_const.h, that specify which IP multicast group to use for the exchange of control information, which port number, default values for protocol parameters (which are overwritten

when `bcast` is started up by XORP, using either the default values in the template file or the entries in the configuration file), and many more. For more information on this header file, see Appendix B.

4. Performance comparisons

We ran experiments with `bcast` on a testbed of Linux laptops with IEEE 802.11a/b/g interface cards. We also compared the protocol performance to our implementation in NS2, described in [Kunz 2003]. As the network interfaces operate in 802.11b mode by default, with a link bandwidth of up to 11 Mbps, we could not use the standard NS2 implementation (which models an 802.11 link at 2 Mbps). Rather, an NS2 MAC layer implementation that can simulate various IEEE 802.11 protocols is provided by INRIA [INRIA 2003, Romdhani 2003]. The main purpose of their work was to explore EDCF, work under study by the IEEE 802.11e working group to support QoS in Wireless LANs [Ni 2002], and to propose enhancements to EDCF in the form of Adaptive EDCF (AEDCF) [Romdhani 2002]. Based on their simulation code, we explored the impact of various MAC data rates on the performance of `bcast` and reliable `bcast`. Certainly, for MAC layers with higher nominal link bandwidth, we expect increased packet delivery ratios.

The simulation results are based on the following set of parameters:

- Area: 500 x 500 meters
- Number of nodes: 6
- Simulation length: 140 seconds
- Number of repetitions: 10
- Physical/Mac layer: IEEE 802.11 DCF, data rates of 2, 11, and 36 Mbps.
- Radio: 250 meter transmission range, 550 meter interference range
- Mobility model: random waypoint model with no pause time, maximum speed either 20 m/s (high mobility scenarios), 1 m/s (low mobility scenarios), or 0 m/s (stationary scenario).
- Traffic patterns: one multicast sender sending to a group of multicast receivers, or a group of multicast senders sending to a single multicast receiver. We have a total of 6 nodes in our network, so either node 0 sends to nodes 1 through 5 or nodes 0 through 4 send to node 5. All receivers join the single multicast group at the beginning of the simulation; the sender(s) start sending data 30 seconds into the simulation. After 130 seconds, all senders stop transmitting data, during the remaining 10 seconds, data packets still in flight have a chance to be delivered. We decided to allow for this additional 10 seconds in our simulations to avoid the problem of how to account correctly for packets that are in flight at simulation end.
- Traffic load: we tested two traffic loads. In low traffic load scenarios, each sender sends 1 packet of size 1024 bytes every second, for a total of 100 packets per sender. In high traffic load scenarios, each sender sends 10 packets of size 1024 bytes per second, for a total of 1000 packets per sender.

The basic performance metrics are Packet Delivery Ratio (PDR) and Packet Latency. Packet delivery ratio is defined as the percentage of received packets, relative to the total number of packets ideally received. Counting the total number of received packets is straightforward in the simulations. The total number of packets ideally received is the number of intended multicast

receivers multiplied by the number of packets sent. Packet latency is the elapsed time between a packet being transmitted and ultimately received. As we did not synchronize the clocks on the laptops during our tests, we will only explore PDR as performance metric, however.

For the testbed measurements, a total of 6 Linux laptops with IEEE 802.11a/b/g LinkSys wireless cards were set up. Each laptop is also connected via Ethernet to each other and the Internet, using a hub. We use MNE [Protean 2004], the Mobile Network Emulator from NRL's PROTocol Engineering Advanced Networking (PROTEAN) Research Group, to emulate mobility scenarios. As described in [MNE 2003], MNE emulates mobile scenarios by exchanging control information about mobile node locations periodically among all hosts. Senders that are deemed out of range will have their packets filtered (rejected) at the MAC layer. MNE can be configured to use a separate connection to exchange control information, reducing its impact on the wireless channel. The example presented in [MNE 2003] indicates the MNE can potentially replicate the behaviour of a real mobile network in some cases. In our testbed, we exchange the MNE control information over the separate Ethernet connection. The following figure depicts this arrangement.

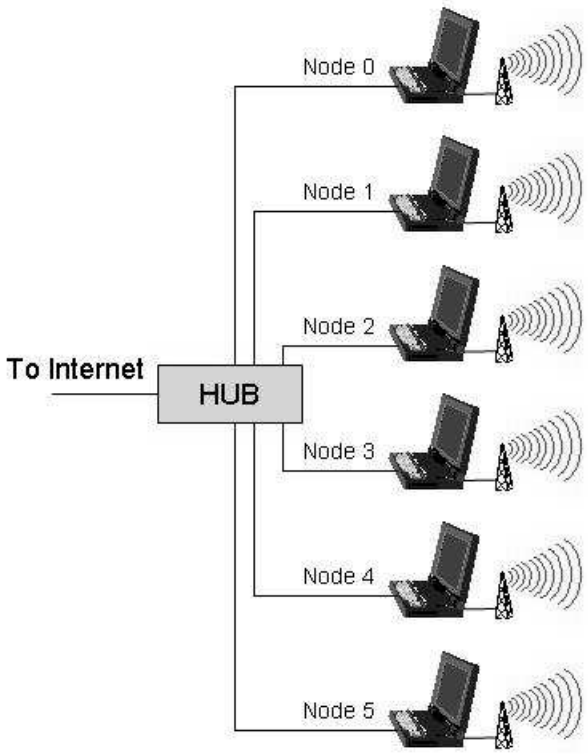

Figure 2: Testbed Configuration

On each laptop, we run either a multicast sender or receiver, based on the traffic pattern. We also run `bcast` and/or MNE, depending on the specific test case. To ensure that the tests are not unduly influenced by other wireless cards in the building, all cards, running in ad-hoc mode, are configured with a unique ESSID. This filters out traffic from neighbouring labs and allows the node to tune in to each other, rather than changing channel and connecting with another laptop that uses the same default ESSID (which we observed a number of times in the beginning).

The presented simulation data is average over 10 runs. For the real measurements, we only repeated each scenario 5 times. In the case of a mobile network, we used the first 5 NS2 scenario files to drive the mobility model in MNE.

The results are presented in a number of ways. The next section summarizes the simulation results for different MAC layer data rates and mobility rates. Then we summarize the case of a static network, where every node is within range of every other node. In this case, we could implement the multicast either by simply broadcasting the multicast packets at the MAC layer (which is trivially implemented by the Linux kernel), we could run an instance of `bcast` on each node (which will increase the network traffic due to the periodic hello messages, and the fact that the protocol may conclude that other nodes may have to retransmit the multicast packet), or by running `bcast` on top of MNE. These combinations allow us to quantify the `bcast` and MNE overhead. In addition, we can run `bcast` in RELIABLE mode, to see whether there is any benefit in that simple case. Finally, in the third section, we will discuss the results under a number of emulated mobility scenarios.

4.1 Simulation results

We simulated three different MAC layer data rates (2 Mbps, 11 Mbps, and 36 Mbps), for both the reliable and unreliable version of `bcast`, under light and heavy traffic (1 packet per second or 10 packets per second per multicast sender). The following table summarizes the results.

Table 1: Simulation Results for MAC Layer data rate of 2 Mbps

PDR		1 Sender, 5 Receiver			5 Sender, 1 Receiver		
		0 m/s	1 m/s	20 m/s	0 m/s	1 m/s	20 m/s
Light Traffic	Unreliable	1.00000	0.82570	0.85590	1.00000	0.86760	0.86969
	Reliable	1.00000	0.82720	0.89470	1.00000	0.86760	0.87880
Heavy Traffic	Unreliable	1.00000	0.82656	0.85764	0.99440	0.84929	0.85321
	Reliable	1.00000	0.82719	0.86114	0.99280	0.85048	0.85554

Table 2: Simulation Results for MAC Layer data rate of 11 Mbps

PDR		1 Sender, 5 Receiver			5 Sender, 1 Receiver		
		0 m/s	1 m/s	20 m/s	0 m/s	1 m/s	20 m/s
Light Traffic	Unreliable	1.00000	0.82630	0.85220	1.00000	0.86780	0.87590
	Reliable	1.00000	0.82760	0.89470	1.00000	0.84940	0.87930
Heavy Traffic	Unreliable	1.00000	0.82672	0.85469	0.99980	0.86707	0.87157
	Reliable	1.00000	0.82729	0.86237	0.99980	0.86663	0.87255

Table 3: Simulation Results for MAC Layer data rate of 36 Mbps

PDR		1 Sender, 5 Receiver			5 Sender, 1 Receiver		
		0 m/s	1 m/s	20 m/s	0 m/s	1 m/s	20 m/s
Light Traffic	Unreliable	1.00000	0.82630	0.85570	1.00000	0.86790	0.87190
	Reliable	1.00000	0.82760	0.89890	1.00000	0.86820	0.87910
Heavy Traffic	Unreliable	1.00000	0.82705	0.85250	1.00000	0.86700	0.87093
	Reliable	1.00000	0.82770	0.86274	1.00000	0.86730	0.87132

These results show a number of trends we would expect, based on our prior studies. All else being equal, a higher MAC layer data rate results in more packets being delivered, in particular for heavy traffic loads. In static networks, where all nodes are within reach of each other, (nearly) all packets are delivered, in scenarios with mobility, the PDR drops substantially. This is due to the random nature of the mobility model, causing network partitions. The packet stream in the two traffic scenarios (1 sender and 5 receivers vs. 5 senders and 1 receiver) is symmetric: all packets either originate from node 5 or are destined to node 5. Therefore, the shape of the network partition does not influence the protocol performance. The reliable protocol in general increases PDR, and that increase is more pronounced under high mobility (which causes partitions to more short-lived) and low traffic loads (which throttle the NACK mechanism less).

It is probably also worth pointing out that the PDR does not decrease for the single receiver case. Even though 5 senders inject more traffic into the network and the traffic all concentrates around the single receiver, potentially causing congestion, the opposite result can be observed in most cases, particularly for high MAC data rates: the PDR is actually higher for this traffic scenario, compared to the case of a single sender. This can be explained as follows: in the single sender scenarios, loosing the original packet due to transmission problems early on causes all 5 receivers to miss that packet. In the case of 5 senders, transmission errors will only impact the delivery of that single packet once. The reliable version of `bcast` can recover from such transmission losses, in particular under light network load, resulting in improved PDR.

4.2 Stationary network

In the case of a static network, with all 6 hosts within transmission range of each other, we can run and compare a number of different tests. Besides the simulation results, we can obtain results by not running `bcast`, running `bcast` but not MNE, and finally by running MNE and `bcast` together. In all cases, we will look at scenarios with either one sender, sending to the remaining 5 nodes, or 5 senders sending to the single remaining node. We look at both low and high traffic loads, and exercise the reliable protocol version as well as the unreliable one. MNE, similar to the NS2 simulations, is configured with static node locations that ensure that all nodes are within communication range of each other.

Table 4: PDR for Various Static Network Cases

PDR		Light Traffic		Heavy Traffic	
		1 Sender	5 Sender	1 Sender	5 Sender
MAC broadcast only		1.00000	0.99880	0.99570	0.99682
Unreliable `bcast`	Without MNE	1.00000	0.99880	0.96884	0.98820
	With MNE	1.00000	0.99920	0.99972	0.97676
Reliable `bcast`	Without MNE	1.00000	0.98840	1.00000	0.97560
	With MNE	1.00000	1.00000	1.00000	0.89584

The simulation results predict that for all scenarios the PDR should be (close to) 100%. This is in general what we observe in most cases. Not surprisingly, given the good performance of the unreliable protocol, the reliable version has little positive impact. However, in the case of 5 multicast senders under heavy load and using MNE, the PDR takes a noticeable hit, dropping to below 90%. We noticed in those experiments that the single multicast receiver received, on average over 40% duplicate packets (i.e., packets sent from the same multicast sender with the same sequence number). This indicates that many retransmission requests flooded the network, which could have resulted in the low overall performance. In the absence of MNE, we did not notice this increase in duplicate packet receptions. While the results were consistent for all 5 repetitions in both cases, it is not clear what caused the different behaviour. One potential explanation could be that the additional packet filtering, when MNE is enabled, adds just enough delay (and delay variation) to packets to prevent the NACK throttle from suppressing NACKs under heavy traffic. However, further studies to explore the impact in particular of the NACK throttle on the protocol performance would be required to answer this question.

Except for the 5 sender scenario under heavy load, MNE seems to have little impact on the protocol performance. `bcast` does tend to negatively impact the protocol performance under heavy traffic with 5 multicast sender, when 5000 packets of size 1 KByte are transmitted in 100 seconds. The additional protocol overhead (periodic HELLO messages, potential for unnecessary re-broadcast of multicast packets) seems to cause network congestion and loss of multicast packets, reducing PDR by 1% to 2% in most cases, except for the special case mentioned above, where PDR dropped by 10%.

4.3 Emulated mobile network

MNE allows us to use an NS2 scenario file and have nodes move around based on the movement described in this file. Together with the radio range, this will cause MNE to either block or enable reception of data from other nodes, based on the relative emulated distance between them. This filtering is done at the MAC layer, using a node's MAC layer address.

Table 5: PDR for Various Mobility Scenarios (Simulation)

PDR		Light Traffic		Heavy Traffic	
		1 Sender	5 Sender	1 Sender	5 Sender
Unreliable `bcast`	Low Speed	0.83460	0.83600	0.83480	0.83572
	High Speed	0.85000	0.86220	0.85344	0.85128
Reliable `bcast`	Low Speed	0.83840	0.83720	0.83594	0.83578
	High Speed	0.89720	0.87060	0.86660	0.85476

Table 6 shows the results we measured in our testbed, averaged over 5 runs. For these runs, we used the first 5 NS2 scenario files. For comparison purposes, Table 5 contains the PDR for the simulation results with an 11 Mbps MAC data rate, averaged over the first 5 simulation runs (and therefore slightly different from the results reported in Table 2, which shows the averages over 10 different scenario files). In the simulation results, the reliable protocol in general increases PDR, and that increase is more pronounced under high mobility (which causes partitions to more short-lived) and low traffic loads (which throttle the NACK mechanism less). Contrary to expectation, the PDR does not decrease for the single receiver case for the unreliable version of the protocol. Even though 5 senders inject more traffic into the network and the traffic all concentrates around the single receiver, the PDR is actually higher, due to the negative effect of loosing a packet transmission from the single sender. The reliable version of `bcast` can recover from such transmission losses, in particular under light network load, resulting in improved PDR for the single-sender cases.

Table 6: PDR for Various Mobility Scenarios (Measurements)

PDR		Light Traffic		Heavy Traffic	
		1 Sender	5 Sender	1 Sender	5 Sender
Unreliable `bcast`	Low Speed	0.95400	1.00000	0.95552	0.87260
	High Speed	0.97520	0.92840	0.92152	0.73844
Reliable `bcast`	Low Speed	0.95440	1.00000	0.95284	0.76084
	High Speed	0.97520	0.93320	0.97688	0.78796

The results for the various mobility scenarios, emulated by MNE and driven by the trace files used for our NS2 experiments, are superficially rather different from the simulation results.

Except for the case of 5 senders under heavy traffic, PDR is in the 92% and higher range. In the latter case, with each of the five sender sending data at high rate, we measured a significant drop in PDR, except for the case of the unreliable protocol under low mobility (PDR of 87.26%). While MNE does filter packets based on MAC addresses, and the MNE tracefiles show that various nodes are out of transmission range over time, clearly more work is required to explore how well MNE models the NS2 simulation scenarios.

Ignoring the actual values, our measurements also cannot confirm the trends we extracted from the simulation runs. Arguably, even in the absence of a complete re-generation of the NS2 scenarios, we should be able to observe similar qualitative trends in our data. However, the reliable version does not increase the PDR for low traffic loads (independent of the mobility rate). While the reliable protocol does, in general, provide at least the same performance as the unreliable version, we achieve the best performance improvement under heavy traffic and high mobility. Under light traffic load, the single-sender scenario has a lower PDR than the multiple-sender scenario, yet the reliable protocol version does not recover from the single-transmitter losses. Under heavy traffic load, the increase in the number of senders caused severe congestion, producing a significant drop in PDR for all mobility rates and protocol versions. A possible explanation could be that while MNE does filter packets at the MAC layer, based on emulated difference, these packets do share the common wireless channel. With 5 senders in close physical but not logical proximity, the packet transmissions and re-transmissions congest the network. Note that this is different from the static network scenarios, where nodes are physically and logically close. In the latter case, other nodes will not re-broadcast packets, leading to a much reduced overall load on the network,

Overall, the results in our actual testbed differ both quantitatively and qualitatively from the results generated by our simulations. Tools like MNE can help in testing and debugging protocols for real testbeds. However, there needs to be more work to explore the suitability of tools such as MNE to evaluate routing protocols qualitatively and quantitatively. For example, it would be interesting to conduct two set sets of experiments with `bcast` in a stationary environment where not all nodes are within reach of each other. In a first set, we would use MNE to emulate the existing and broken links, in the second set of runs we would physically distribute the nodes and just run the protocol over the wireless links. We also could re-run measurements for the same mobility scenario, exploring whether random outside interference with the wireless channel and the unpredictable packet scheduling delays at the device driver and the process scheduling within the Linux kernel have an impact on our results.

5. Conclusions and future work

The results show that overall `bcast` performs well. In the vast majority of cases, PDR reaches 99% or higher in the stationary environment and beats the simulation predictions in the mobile scenarios (though the latter may more be related to our use of MNE rather than the actual protocol performance). The protocol imposes no noticeable overhead in the static scenarios, compared to simply using MAC broadcasts, and is essential for multi-hop wireless communication. MNE had little noticeable impact as well, mostly because we provided a separate control connection over which to exchange its control information. For mobile scenarios, the results in our testbed differ both quantitatively and qualitatively from the results generated by simulations. Tools like MNE can help in testing and debugging protocols for real testbeds. However, it is not clear whether tools such as MNE are suitable to evaluate routing protocols qualitatively and quantitatively.

Using XORP as implementation platform for `bcast` was overall not that advantageous. It does provide for a more uniform way to configure the protocol via the configuration file and `rtrmgr`. In the future, once the I/O facilities in FEA are better developed, the protocol communication (HELLO and NACK messages) could be ported to use the appropriate FEA XRLs. This would simplify porting `bcast` to other platforms that support XORP. However, `bcast` will always be somewhat platform-specific, as it has to capture all multicast data packets, selectively re-broadcasting some. We use PCAP in our implementation, and as yet there is no indication the XORP is planning to provide such functionality in a portable way.

Future work can proceed in a number of directions. Clearly more work is necessary to explore the difference in protocol behaviour using simulation and when emulating a wireless environment with MNE. Also, more in-depth analysis of the protocol behaviour under a number of scenarios, in particular ones with higher traffic load, should be done. The results also show that the NACK throttle may have to be tuned appropriately for the MAC layer we use (the parameter settings were derived experimentally using NS2 and a 2 Mbps MAC). Finally, the protocol itself could be enhanced/complemented in at least two ways:

- As already indicated by the simulation results reports in [Kunz 2003], a flow control mechanism could be beneficial to reduce packet loss due to network congestion.

- The protocol is not secure. Nodes could advertise many other nodes as their neighbours, which can prevent these nodes from receiving data (as other nodes now wrongly believe that these nodes already received the broadcast originating from the intruder node). Ways to secure `bcast` could center either around cryptographically securing the HELLO messages or to analyze the HELLO message content to detect anomalies and ignore information from suspicious neighbours. The first solution requires a protocol to distribute keys to encrypt/authenticate HELLO messages, the second approach trades off the protocol optimization for increased robustness of operation in the presence of intruders.

References

[Cheng 2001] E. Cheng, *On-demand Multicast Routing in Mobile Ad Hoc Networks,* M.Eng. Thesis, Carleton University, Ontario, Canada. January 2001.

[Click 2003] The Click Modular Router Project, http://www.pdos.lcs.mit.edu/click/

[Ding 2002] W. Ding, *Multicast Routing in Fixed Infrastructure and Mobile Ad Hoc Wireless Networks with a Multicast Gateway,* M.Sc. (ISS) Thesis, Carleton University, Ontario, Canada, July 2002.

[Grudin 2002] J. Grudin, *Group Dynamics and Ubiquitous Computing*, Communications of the ACM, pp. 74-78, Vol. 45, No. 12, December 2002.

[INRIA 2003] Adaptive EDCF NS2 code, http://www-sop.inria.fr/planete/qni/Research/AEDCF/

[Kunz 2002] T. Kunz, *Multicasting: From Fixed Networks to Ad Hoc Networks*, in Handbook of Wireless Networks and Mobile Computing, pp. 495-507, John Wiley & Sons 2002, ISBN 0-471-41902-8.

[Kunz 2003] T. Kunz, *Reliable Multicasting in MANETs*, Contractor Report, Communications Research Centre, Ottawa, Canada, July 2003.

[Lee 2000] S.-J. Lee, W. Su, J. Hsu, M. Gerla, and R. Bagrodia, *A Performance Comparison Study of Ad Hoc Wireless Multicast Protocols*, in Proc. of the 19th Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM 2000), Vol. 2, pp. 565-574, Tel Aviv, Israel, Mar. 2000.

[Li 2002] V. Li and Z. Zhang, *Internet Multicast Routing and Transport Control Protocols*, in Proc. of the IEEE, pp. 360-391, Vol. 90, No. 3, March 2002.

[Lou 2002] W. Lou and J. Wu, *On Reducing Broadcast Redundancy in Ad Hoc Wireless Networks*, IEEE Transactions on Mobile Computing, pp. 111-122, Vol. 1, No. 2, April 2002.

[MNE 2003] W. Chao, J. P. Macker, and J. W. Weston, *NRL Mobile Network Emulator*, Technical Report NRL/FR/5523--03-10,054, Naval Research Lab, Washington, USA

[Ni 2002] Qiang Ni, Lamia Romdhani, Thierry Turletti and Imad Aad. *QoS Issues and Enhancements for IEEE 802.11 Wireless LAN*. INRIA Research Report No. 4612, Nov. 2002.

[Obraczka 2001a] K. Obraczka, G. Tsudik, and K. Viswanath, *Pushing the Limits of Multicast in Ad Hoc Networks,* in Proc. of the 21st Int. Conference on Distributed Computing Systems, pp. 719-722, Phoenix, USA, 2001.

[Obraczka 2001b] K. Obraczka, K. Viswanat, and G. Tsudik, *Flooding for Reliable Multicast in Multi-Hop Ad Hoc Networks,* Wireless Networks, pp. 627-634, Vol. 7, No. 6*,* 2001.

[Pcap 2003] The TCPDUMP public repository, http://www.tcpdump.org/

[Protean 2004] PROTocol Engineering Advanced Networking (PROTEAN) Research Group, Navy Research Lab, USA, http://protean.itd.nrl.navy.mil/

[RFC 1700] J. Reynolds, J. Postel, *Assigned Numbers*, October 1994. (Status: HISTORIC)

[RFC 2357] A. Mankin, A. Romanow, S. Bradner, V. Paxson, *IETF Criteria for Evaluating Reliable Multicast Transport and Application Protocols*, June 1998. (Status: INFORMATIONAL)

[RFC 2375] R. Hinden, S. Deering, *IPv6 Multicast Address Assignments*, July 1998. (Status: INFORMATIONAL)

[RFC 2908] D. Thaler, M. Handley, D. Estrin, *The Internet Multicast Address Allocation Architecture*, September 2000. (Status: INFORMATIONAL)

[RFC 3170] B. Quinn, K. Almeroth, *IP Multicast Applications: Challenges and Solutions*, September 2001. (Status: INFORMATIONAL)

[RFC 3171] Z. Albanna, K. Almeroth, D. Meyer, M. Schipper, *IANA Guidelines for IPv4 Multicast Address Assignments*, August 2001. (Status: BEST CURRENT PRACTICE)

[RFC 3306] B. Haberman, D. Thaler, *Unicast-Prefix-based IPv6 Multicast Addresses*, August 2002. (Status: PROPOSED STANDARD)

[RFC 3307] B. Haberman, *Allocation Guidelines for IPv6 Multicast Addresses*, August 2002. (Status: PROPOSED STANDARD)

[Romdhani 2002] Lamia Romdhani, Qiang Ni, and Thierry Turletti. *AEDCF: Enhanced Service Differentiation for IEEE 802.11 Wireless Ad-Hoc Networks*, INRIA Research Report No. 4544, 2002.

[Romdhani 2003] Lamia Romdhani, Qiang Ni, and Thierry Turletti. *Adaptive EDCF: Enhanced Service Differentiation for IEEE 802.11 Wireless Ad Hoc Networks*. IEEE WCNC'03 (Wireless Communications and Networking Conference), New Orleans, Louisiana, March 2003.

[Varshney 2002] U. Varshney, *Multicast over Wireless Networks*, Communications of the ACM, pp. 31-37, Vol. 45, No. 12, December 2002.

[Williams 2002] B. Williams and T. Camp, *Comparison of Broadcast Techniques for Mobile Ad Hoc Networks*, in Proc. of the Third Symposium on Mobile Ad Hoc Networking and Computing, pp.194-205, Lausanne, Switzerland, June 2002.

[Xorp 2003] eXtensible Open Routing Platform Project, http://www.xorp.org/

[Zhu 2002] Y. Zhu, *Pro-Active Connection Maintenance in AODV and MAODV,* M.Sc. (ISS) Thesis, Carleton University, Ontario, Canada, August 2002.

Appendix A: XORP

The XORP project [Xorp 2003] is developing an open source software router. The software is intended to be stable and fully featured enough for production use, and flexible and extensible enough for research use. As of release 0.5 from November 2003 (the version our work is based upon), XORP runs natively on FreeBSD and Linux, with support for other platforms and the Click Modular Router [Click 2003] expected in the future. The bulk of this section is taken from the software design documents available at the Xorp project website, `www.xorp.org`.

The goal for XORP is to become a suitable software platform that can be used as the core of practically any router. However, the *initial* focus is on an edge router running on commodity PC hardware with relatively low port density and moderate size routing tables. The XORP design philosophy stresses *extensibility*, *performance* and *robustness*. For routing and management modules, the primary goals are extensibility and robustness. These goals are achieved by carefully separating functionality into independent modules, running in separate Unix processes, with well-defined APIs between them. Clearly, there are performance penalties to pay for such an architecture, but so long as careful attention is paid to computational complexity, the costs associated with inter-process communication will be acceptable given the obvious extensibility and robustness benefits. For the forwarding path, the primary goals are extensibility and performance. Robustness here is primarily achieved through simplicity and a modular design that encourages re-use of well-tested components.

XORP can be divided into two subsystems. The higher-level ("user-space") subsystem consists of the routing protocols and management mechanisms. The lower-level ("kernel") provides the forwarding path, and provides APIs for the higher-level to access. User-level XORP uses a multi-process architecture with one process per routing protocol, and a novel inter-process communication mechanism known as XORP Resource Locators (XRLs). XRL communication is not limited to a single host, and so XORP can in principle run in a distributed fashion. The lower-level subsystem can use traditional UNIX kernel forwarding, or the Click modular router, as mentioned above. The modularity and minimal dependencies between the lower-level and user-level subsystems allow for many future possibilities for forwarding engines.

Figure A.1 shows the processes in XORP, although it should be noted that some of these modules use separate processes to handle IPv4 and IPv6. For simplicity, the arrows show only the main communication flows used for routing information. Control flows are not shown - for example, the FEA may need to inform the routing processes when an interface goes down. It is also worth pointing out the even though the design philosophy is that each XORP component in Figure 1 will run as a separate process, it would also be possible to compile most of them together to run as one single process. Obviously, the robustness of such a router would suffer because the crash of a single component would bring down the whole router. However, the XORP architecture is designed to be flexible, and other developers building on this software can choose to use it in different ways.

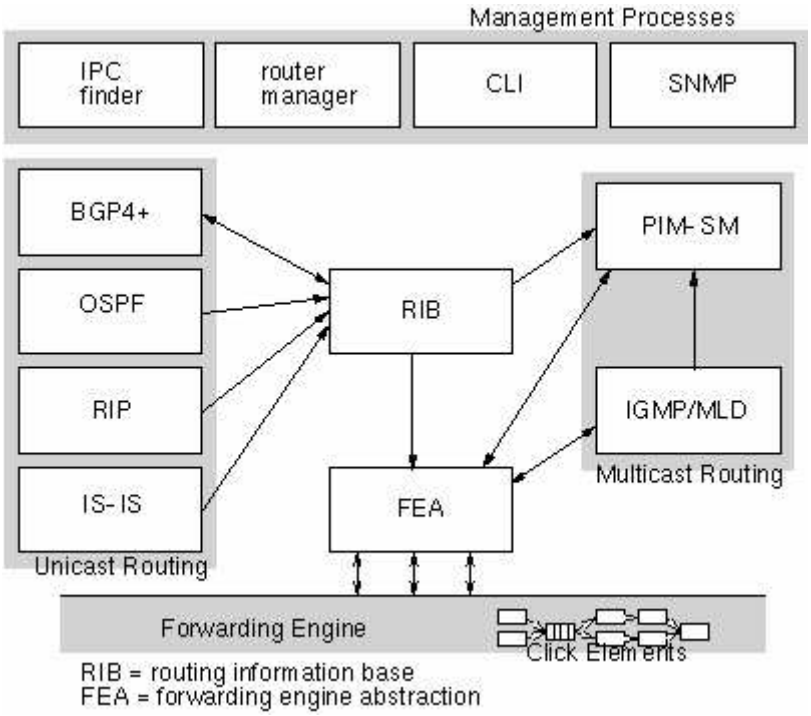

Figure A.1: Xorp Architecture

The key components are the FEA and RIB. The FEA (Forwarding Engine Abstraction) provides a uniform interface to the underlying forwarding engine. It shields XORP processes from concerns over variations between platforms. As a result, XORP processes need not be concerned whether the router is comprised of a single machine, or cluster of machines; or whether the network interfaces are simple, like a PCI Ethernet adapter, or are smart and have processing resources, like an Intel IXP cards. The FEA performs three distinct roles: *interface management*, *forwarding table management*, and *interface-specific packet I/O*. The RIB (Routing Information Base) holds a user-space copy of the entire routing/forwarding table, complete with information about where each route came from (*e.g.,* which protocol, and when). It communicates with the routing protocols such as BGP, RIP and OSPF to instantiate routes, and with the FEA to install the appropriate forwarding entries in the FEs.

XORP comes with implementations of a number of standard routing protocols. For research purposes, these processes (and all others comprising the router) may be started manually or from scripts, so long as the dependencies between then are satisfied. But when using XORP in a more operational environment, the network manager typically does not wish to see the software structure, but rather would like to interact with the router *as a whole*. Minimally, this consists of a configuration file for router startup, and a command line interface to interact with the router during operation. The `rtrmgr` process provides this unified view of the router.

The `rtrmgr` is normally the only process explicitly started at router startup. The rtrmgr process includes a built-in XRL finder, so no external finder process is required. The following sequence of actions then occurs:

1. The `rtrmgr` reads all the template files in the router's template directory. Typically there is one template file per XORP process that might be needed. A template file

describes the functionality that is provided by the corresponding process in terms of all of the configuration parameters that may be set. It also describes the dependencies that need to be satisfied before the process can be started. After reading the template files, the `rtrmgr` knows all the configuration parameters currently supportable on this router, and it stores this information in its *template tree*.

2. The `rtrmgr` next reads the contents of the XRL directory to discover all the XRLs that are supported by the processes on this router. These XRLs are then checked against the XRLs in the template tree. As it is normal for the XRLs in the XRL directory to be used to generate stub code in the XORP processes, this forms the definitive version of a particular XRL. Checking against this version detects if a template file has somehow become out of sync with the router's codebase. Doing this check at startup prevents subtle run time errors later. The `rtrmgr` will exit if a mismatch is discovered.

3. The `rtrmgr` then reads the router configuration file. All the configuration options in the config file must correspond to configurable functionality as described by the template files. As it reads the config file, the `rtrmgr` stores the intended configuration in its *configuration tree*. At this point, the nodes in the configuration tree are annotated as *not existing* - that is this part of the configuration has not yet been communicated to the process that will implement the functionality.

4. The `rtrmgr` next traverses the configuration tree to discover the list of processes that need to be started to provide the required functionality. Typically not all the available software on the router will be needed for a specific configuration.

5. The `rtrmgr` traverses the template tree again to discover an order for starting the required processes that satisfies all their dependencies.

6. The `rtrmgr` starts the first process in the list of processes to be started.

7. If no error occurs, the `rtrmgr` traverses the configuration tree to build the list of XRLs that need to be called to configure the process just started. These XRLs are then called, one after another, with the successful completion of one XRL triggering the calling of the next. Some processes may require calling a transaction start XRL before configuration, and a transaction complete XRL after configuration - the `rtrmgr` can do this if required.

8. If no error occurred during configuration, the next process is started, and configured, and so forth, until all the required processes are started and configured.

9. At this point, the router is up and running. The `rtrmgr` will now allow connections from the `xorpsh` process to allow interactive operation.

Appendix B: Relevant `bcast` files

`bcast` implements the XRL interface shown in Figure B.1.

```
/*
 * Interface definition for the bcast routing protocol.
 * August 2003, Thomas Kunz.
 */
interface bcast/1.0 {
        /**
         * Set HELLO interval (in seconds)
         * @param new_int new HELLO interval
         */
        set_hello_interval ? new_int:i32

        /**
         * Get HELLO interval
         * @param old_int current HELLO interval
         */
        get_hello_interval -> old_int:i32

        /**
         * Set buffer size
         * @param new_buf size of packet buffer
         */
        set_buffer_size ? new_buf:i32

        /**
         * Get buffer size
         * @param old_buf current packet buffer size
         */
        get_buffer_size -> old_buf:i32

        /**
         * Set RELIABILITY (TRUE or FALSE)
         * @param reliability set reliablity level for bcast
         */
        set_reliability ? reliability:bool

        /**
         * Get RELIABILITY setting (TRUE or FALSE)
         * @param reliability get  reliability level for bcast
         */
        get_reliability -> reliability:bool

        /**
         * Set interface over which bcast operates
         * @param intf set interface over which bcast operates
         */
        set_interface ? intf:txt

        /**
         * Get interface over which bcast operates
         * @param intf get interface over wich bcast operates
         */
        get_interface -> intf:txt

        /**
         * Activate the protocol
         */
        activate_bcast

}
```

Figure B.1: `bcast` Interface Definition

The router manager `rtrmgr` reads a directory of template files to discover the configuration options that the specific protocol or module supports. Figure B.2 shows the `bcast` template.

```
protocols {
  bcast {
    hello-int: uint = 1;    /* by default, send hello message every second */
    buffer-size: uint = 10; /* by default, buffer last 10 packets */
    reliability: bool=false;/* by default, run best-effort version */
    proto_intf: text = "lo";/* use loopback device by default */
  }
}

protocols {
  bcast {
    %modinfo: provides bcast;
    %modinfo: path "bcast/xorp_bcast";
    %modinfo: statusmethod xrl;
    %modinfo: shutdownmethod xrl;
    hello-int {
      %set: xrl "bcast/bcast/1.0/set_hello_interval?new_int:i32=$(@)";
      %get: xrl "bcast/bcast/1.0/get_hello_interval->old_int:i32";
    }
    buffer-size {
      %set: xrl "bcast/bcast/1.0/set_buffer_size?new_buf:i32=$(@)";
      %get: xrl "bcast/bcast/1.0/get_buffer_size->old_buf:i32";
    }
    reliability {
      %set: xrl "bcast/bcast/1.0/set_reliability?reliability:bool=$(@)";
      %get: xrl "bcast/bcast/1.0/get_reliability->reliability:bool";
    }
    proto_intf {
      %set: xrl "bcast/bcast/1.0/set_interface?intf:txt=$(@)";
      %get: xrl "bcast/bcast/1.0/get_interface->intf:txt";
    }
    %activate: xrl "bcast/bcast/1.0/activate_bcast";
  }
```

Figure B.2: `bcast` Template Definition

The template file consists of two components. The first part specifies the configurable options and default values for them. In addition to specifying the configurable options, the template file should also specify what the rtrmgr should do when an option is modified. These commands annotating the template file begin with a "`%`".The first three annotations apply to the "protocols bcast" node, and specify the "%modinfo" command, which provides information about the module providing this functionality. In this case they specify that this functionality is provided by the module called `bcast` and that the executable for this module is `bcast/xorp_bcast`. For more details on the syntax of either the interface definition file or the template file, see [Xorp 2003].

The main idea behind the protocol is to intercept multicast packets and selectively re-broadcast them. To identify duplicate and missing multicast packets, a multicast sender has to prefix each multicast packets with the bcast protocol header, in addition to the IP and UDP header, see Figure B.3. This header is also used by the protocol itself to exchange control information, the type field distinguishes the three supported packet types

```
struct bcastHeader {
  u_int16_t     type;
  in_addr_t     sender_addr;
  u_int32_t     sequence_no;
  in_addr_t     prev_hop;
};

// The following three message types are supported:
#define bcast_DATA_PACKET       1
#define bcast_HELLO_MSG         2
#define bcast_NACK              3
```

Figure B.3: `bcast` Protocol Header and Constants

The applications `bcastSender` and `bcastReceiver` demonstrate the use of this protocol header. `bcastSender` plays the role of a multicast sender, expecting 4 command-line parameters: the IP multicast group, the time (in ms) between successive multicast packets, the total number of multicast packets to be send, and the size of the multicast packet (excluding the header). The sender will issue a sequence of packets of fixed size and with fixed interarrival times, filling in the header fields with its own address in both `sender_addr` and `prev_hop`, and an increasing sequence number, starting from 0 for the first packet, in `sequence_no`. `bcastReceiver` expects one command-line parameter, the IP multicast address, and will block listening for IP packets send to this address. Every time a packet is received, `bcastReceiver` will print out the relevant information from the `bcast` protocol header.

Figure B.4 reproduces the protocol header file, which defines and comments widely used constants in the protocol implementation.

```
/*
 * Constants defined for bcast protocol
 */

#ifndef bcast_const_hh

/* each instance of bcast is uniquely identified by an ID. We
 * choose to make this the IP address of the main interface.
 * which has name bcast_INTERFACE.
 */

#define bcast_INTERFACE "eth0"

/* constants to propagate a bcast CONTROL message to all neighbours,
 * using multicasting over one hop (all nodes register with the
 * well-known multicast group to exchange control information...)
 */

#define bcast_GROUP             "224.0.0.128" // not currently assigned by IANA
#define bcast_PORT              1500          // more or less arbitrary, > 1023
#define bcast_TTL               1             // propate ONE hop
#define bcast_MAX_MSG           2000          // hopefully never more

/* the main loop is event-driven, so periodically poll the various
 * file descriptors whether new data (control message or user data)
 * is available to be read in, preventing BLOCKING I/O.
 */

#define bcast_POLL_PERIOD       20    // poll every 20 milliseconds

/* some other events get scheduled periodically: clean up list of
 * unique packet IDs, check for expired neighbors, etc. The periodicity
 * of these events is defined by the following constants.
 */

#define CLEANUP_PERIOD  5000         // every 5000 ms
```

```
// how much to delay packet retransmission (scaling factor)
#define bcast_DELAY     100          // in ms

// how much time to delay NACK'd packet retransmissions (scaling factor)
#define PENDING_DELAY   10           // in ms

/* we use NACKs up to MAX_NACK times for a specific packet that is deemed
 * lost. After trying for MAX_NACK times, we will give up. The timer is
 * cancelled when the requested packet is received.
 */

#define MAX_NACK 3
#define NACK_INITIAL_DELAY 10   // upper bound (in ms) on initial NACK jitter
#define NACK_TIMEOUT 1000               // timeout and try again after x ms

// constants to control frequency of NACKs (the throttling mechanism)
#define MIN_DIFF        700000  // do not send too many NACKs per unit time
#define PACKET_COUNT    30      // if we exceed this, do not issue NACK

/* We periodically send HELLO messages to neighbors. To avoid HELLO
 * message collisions, they are randomly scheduled to occur between
 * [0.75 * HELLO_INTERVAL, 1.25 * HELLO_INTERVAL]. If we miss more than
 * ALLOWED_HELLO_LOSS messages from the neighbors, we assume the neighor
 * is gone.
 */

#define DEFAULT_HELLO_INTERVAL  2             // in seconds
#define ALLOWED_HELLO_LOSS      1             // packets
#define MaxHelloScaling         1.25
#define MinHelloScaling         0.75

/* we also buffer user data in a local buffer, a buffer whose size
 * is determined by configuration parameters
 */

#define DEFAULT_BUFFER_SIZE     10            // at most last 10 packets

#endif
```

Figure B.4: `bcast` Implementation Constants

List of symbols/abbreviations/acronyms/initialisms

DND	Department of National Defence
ACK	Acknowledgement
AEDCF	Adaptive Enhanced Distributed Coordination Function
AODV	Ad-hoc On-demand Distance Vector
ARQ	Automatic Repeat reQuest
bcast	BroadCAST
DCF	Distributed Coordination Function
DRDC	Defence Research and Development Canada
EDCF	Enhanced Distributed Coordination Function
ID	IDentifier
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
INRIA	Institut National de Recherche en Informatique et en Automatique
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
MAC	Medium Access Control
MANET	Mobile Ad-hoc NETwork
MAODV	Multicast extensions for AODV
MBone	Multicast Backbone
Mbps	Megabits per second
MNE	Mobile Network Emulator
ms	milliseconds
NACK	Negative ACKnowledgement

NS2	Network Simulator version 2
ODMRP	On-Demand Multicast Routing Protocol
PDR	Packet Delivery Ratio
RFC	Request For Comment
TCP	Transmission Control Protocol
TTL	Time-To-Live