Improving Link Failure Detection and Response in IEEE 802.11 Wireless Ad hoc Networks

Alvin Valera[1] Hwee-Pink Tan[1] Winston K.G. Seah[2]

[1]Institute for Infocomm Research (I2R)
1 Fusionopolis Way, #21-01 Connexis, Singapore 138632
E-mail: {acvalera,hptan}@i2r.a-star.edu.sg

[2]School of Engineering and Computer Science, Victoria University
PO Box 600, Wellington 6140, New Zealand
E-mail: Winston.Seah@ecs.vuw.ac.nz

***Abstract*—Wireless multihop ad hoc networks face a multitude of challenging problems including highly dynamic multihop topologies, lossy and noisy communications channels, and sporadic connectivity which contribute to frequent link failures. Rapid and accurate link failure detection is therefore important to maintain correct and optimum operation of network routing protocols. In this paper, we propose a unified link failure detection and recovery architecture (`ulfra`) which uses link layer feedback for rapid failure detection and packet salvaging for packet recovery. While link layer feedback and packet salvaging have been studied in simulations and simple experiments, no thorough experimental study have been undertaken to evaluate their real-world performance. This paper essentially fills this void as we implement and evaluate `ulfra` in an IEEE 802.11 multihop ad hoc network. Our experimental results show that link layer feedback, as modeled in current network simulators, actually performs worse than hello beaconing as it generates excessive false failure detections. To improve its performance, we implement a *veto mechanism* to reduce spurious detections. Experimental results show that the veto mechanism dramatically improves the performance of link layer feedback in terms of packet delivery, delay, and routing overhead as it considerably reduces the number of false detections. Compared with hello, it delivers 15–20% more packets at high node failure and 12–20% more at high network traffic.**

I. INTRODUCTION

Wireless multihop ad hoc networks face a multitude of challenging problems including highly dynamic multihop topologies, lossy and noisy communications channels, and sporadic connectivity [1], [2]. These factors contribute to frequent link failures which severely degrade the performance of network protocols. Given that link failures are more the norm than exception, rapid and accurate failure detection is therefore important to ensure correct and optimum network operation.

In wired networks, rapid link failure detection is given primary importance specially in mission-critical deployments where stringent service level agreements must be guaranteed [3]. And while wired networks are aiming for sub-50 milliseconds failure detection and recovery in real deployments [3], no similar proposal has ever been presented to improve link failure detection in wireless ad hoc networks. The best that has been accomplished are simple experimental studies [4], [5] and numerous simulation-based studies [6], [7] on failure detection using *hello beaconing* or *link layer feedback*. In terms of real implementations, routing protocols employ hello-based failure detection which is slow and suffers from many serious problems [4].

In this paper, we introduce a unified link failure detection and recovery architecture (`ulfra`) to address the deficiencies in existing implementations. The cross-layer architecture is routing protocol-independent and employs two well-known techniques in ad hoc networks [8], [9]: link layer feedback to improve failure detection and packet salvaging as a recovery mechanism. We implement the architecture in Linux and evaluate its performance in an IEEE 802.11-based wireless multihop ad hoc network. While link layer feedback and packet salvaging have been studied and used in simulations, their real-world performance remain unknown.

Our experimental results reveal that in reality, link layer feedback as modeled in widely-used network simulators [10], [11], does not necessarily provide improvement. In the case of link layer feedback which uses unicast packets as link probes, it generates considerable false detections when the packet sending rate is high and short-term link quality variations occur. To minimize spurious detections, we implement a *veto mechanism* into the basic link layer feedback. Our results show that the veto mechanism dramatically improves the performance of link layer feedback as it considerably reduces false detections. Its packet delivery ratio (PDR) is 15–20% and 12–20% more than that of hello at high node failure and network traffic rates, respectively.

The remainder of this paper is organized as follows. Section II presents an overview on link failure detection and related work. Section III presents the design and implementation of `ulfra`. Section IV discusses the performance evaluation results, and Section V concludes the paper with a summary of the important findings and future work.

II. LINK FAILURE DETECTION AND RELATED WORK

In wireless ad hoc networks, link failure detection mechanism can be performed using either *periodic hello beacons* [4], [13] or *link layer feedback* [8], [9]. Table I summarizes some relevant studies on failure detection. While there have

TABLE I
WIRELESS AD HOC NETWORK LINK FAILURE DETECTION STUDIES

Authors	Failure Detection Studied	Methodology
Broch *et al*, 1998 [6]	LLF	Simulations
Chakeres & Royer, 2002 [4]	Hello	Experiments
Wang *et al*, 2005 [7]	Hello	Simulations
Huang *et al*, 2006 [12]	Hello	Simulations
Owada *et al*, 2007 [5]	Hello and LLF	Experiments

been numerous papers on this topic, most of them have been conducted using simulations. The experimental studies by Chakeres and Royer [4] and Owada *et al.* [5] were simple, unrealistic, and did not thoroughly investigate the performance of link layer feedback. As such, none of these efforts managed to discover the severe problem of link layer feedback discussed in Sections IV-A and IV-B.

The use of keep-alive packets (referred to as *hello*) for link status monitoring has its origins in wired networks.It has been adapted to wireless networks, in particular to wireless ad hoc networks where it has been used by many routing protocols for maintaining local connectivity [13], [14].

As the characteristics of wireless links differ significantly from that of wired links, numerous problems arose with the use of hello beacons in wireless ad hoc networks. Link layer measurement studies show that wireless links are generally lossy in nature [2] and that the loss rate is affected by several factors including data rate, transmit power, noise, multi-path, RF interference, and packet size [2], [15]. Links with non-negligible loss rates affect the use of hello beacons in terms of false link breakage detections.

The use of link layer acknowledgement to detect link failures was proposed in on demand routing protocols, particularly DSR [6]. The feature exploits the widely-used IEEE 802.11 standard [16] as it provides ACKs for unicast packets. Subsequently, various protocols (e.g. [9]) have incorporated the use of link layer feedback for link breakage detection. Unfortunately, due to the difficulty in implementing link layer feedback (because of the need to modify device drivers), existing routing protocol implementations still rely on hello for detecting link failures [17]–[19]. As such, detection delay in current implementations range in the order of seconds.

Packet recovery mechanisms have been proposed but are tightly coupled with routing protocols. DSR [8] proposed the use of *packet salvaging* wherein affected packets are purged from the interface transmit queue and re-routed if alternative routes are available. CHAMP [9] extended this idea further by incorporating *data caching* to enable distributed packet salvaging. `ulfra` was designed to decouple this recovery function from routing protocols. In this manner, routing protocols can concentrate on their core function of building and maintaining a consistent routing table. We have shown that packet salvaging can indeed improve delivery but worsen delay in real-world ad hoc networks.

Kawadia *et al.* [20] proposed an extension to the routing architecture to address the requirements of on-demand routing. Their extension incorporates “on-demand routing component” (ODRC) which implements the basic on-demand routing functionalities such as informing the user-space daemon of packets requiring route discovery, refreshing used routes, and removing stale routes. `ulfra` is different from ODRC as it focuses on rapid link failure detection and packet recovery.

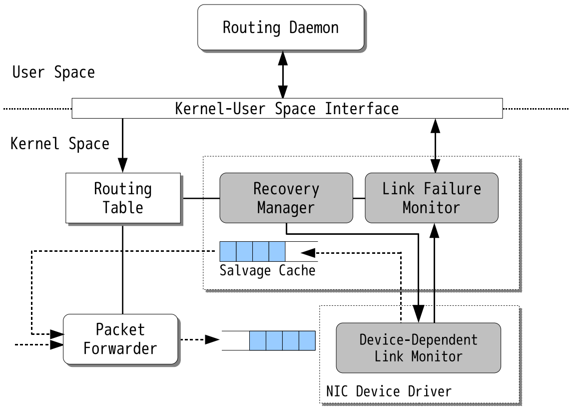


Fig. 1. Unified link failure response architecture. The unshaded blocks are part of the current routing architecture.

III. UNIFIED ARCHITECTURE

We present the design and implementation of a *unified link failure detection and response architecture* (`ulfra`) that will address the shortcomings of the current routing architecture.

A. Design

The main design goal of `ulfra` is to support link layer feedback and salvaging of post-routed packets that are affected by link failures. Figure 1 shows `ulfra` and how it interacts with the current routing architecture. `ulfra` is divided into a device-independent module and a device-dependent module. The former consists of a *link failure monitor*, *recovery manager*, and *salvage cache* while the latter is patched into the device driver and has three low-level functions: (*i*) to detect link failures by tapping into the low-level functions provided by the device; (*ii*) to purge the device transmit queue of affected packets; and (*iii*) to en-queue the purged packets into the salvage cache.

The link failure monitor can receive failure notifications from several devices. When it receives a notification, it immediately informs the routing daemon and recovery manager of the link failure. The recovery manager then invokes the device-dependent module to purge the interface queue and enqueue affected packets in the salvage cache. The recovery manager also checks if the salvaged packets can be re-routed (*i.e.*, if there is alternative route in the routing table). It also monitors the routing table for modifications. If routes are added, it checks if packets in the salvage cache can be re-routed.

The lifetime of the packets in the salvage cache is controlled by the routing daemon. When a link failure occurs, the routing protocol may perform route repair or a new route discovery. When it fails to find an alternative route, it must purge the salvage cache accordingly.

B. *Implementation*

We implemented `ulfra` on the Mikrotik RB-433 router platform which runs the Linux operating system. It is equipped with a wireless LAN card that uses the Atheros AR5212 chipset. The router board itself has an Atheros AR7130 CPU (MIPS) that operates at 300 MHz. The system has 32 MB of main memory and 64 MB of flash storage space. We chose the Atheros chipset as the driver source codes are publicly available in an open-source project called `madwifi` [21]. We replaced the Mikrotik firmware with the OpenWRT distribution (Kamikaze version 8.09). Controlling the various parameters of the wireless interface card was accomplished with the use of `wireless-tools` package.

1) Device-Independent `ulfra` *Module:* We implemented the device-independent `ulfra` module as a stand-alone loadable kernel module in Linux (version 2.6). Netlink socket facilities were used to communicate with user-space applications. The architecture required modifications to the Linux network stack to allow direct access to the packet forwarder but this was avoided as it may cause the system to become unstable. Instead, the implementation used the `netfilter` hooks to re-inject packets back into the network stack.

2) Device-Dependent `ulfra` *Module:* While implementing the device-independent `ulfra` module was straightforward, implementing the device-dependent `ulfra` module into the Atheros madwifi driver (revision r3314) was more challenging. We implemented the required functionalities into the `if_ath.c` source file. This file has a function named `ath_tx_processq()` that is invoked after the hardware completed processing the packets in its transmit buffers. For every packet in the buffer, the hardware marks whether it has been successfully transmitted or not. When a packet is marked as not successfully transmitted due to excessive retransmission attempts, the device-independent `ulfra` module is immediately notified about the failure by invoking the interface function. This approach mimics the link layer feedback model implemented in simulators such as ns-2 [10] and Qualnet [11].

3) Routing Protocol: To complete the implementation, an ad hoc routing protocol needs to be integrated into the architecture. For this purpose, we chose the AODV [13] implementation from the University of Uppsala [17]. The AODV daemon was modified to communicate with `ulfra` through a netlink socket. Whenever a link failure notification is received by the daemon, the function `neighbor_link_break()` is invoked. This function removes the link to the node that failed and all the other routes that pass through the failed link.

IV. EXPERIMENTAL EVALUATION

In this section, we present the performance evaluation of `ulfra` using experimentation. We used all the six routers shown in Figure 2 to establish the topology shown in Figure 3. By ensuring the existence of a path from node 1 to node 6 at all times, we are ensuring that losses are mainly due to failure detection delay and not due to path unavailability. Similarly, the provision of high quality links (by fixing the data rate at 11Mbps and transmit power at 18dBm) ensures that losses not due to wireless transmission errors.

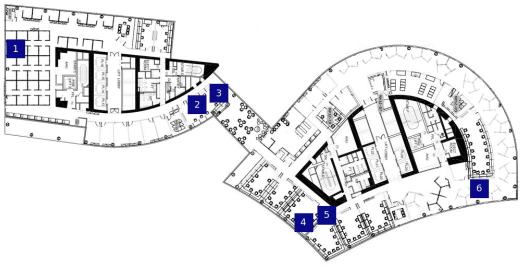


Fig. 2. Indoor testbed deployment map.

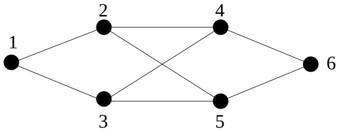


Fig. 3. Topology for performance evaluation of link layer feedback and hello

In the experiments, node 1 was configured to send CBR traffic to node 6 for 300 seconds. Packet size was fixed at 1400 bytes while the sending rate was varied from 25 to 200 packets per second (pkt/sec). The nodes 2–5 are made to fail alternately by switching the channel of the ”failed” node and reducing its transmit power to 1dBm. The rate of failure was varied from 0 (no failure) to 4 (each node failed 4 times every minute.) We used $K = 2$ (number of consecutive hello packets lost before link is marked as down) and $L = 7$ (MAC retry limit) as they provided balanced performance for hello and link layer feedback, respectively, in our preliminary optimizations. Each experiment was repeated fourteen times and we present the average *PDR*, *end-to-end (e2e) delay*, *routing overhead* and *false link failure detections*.

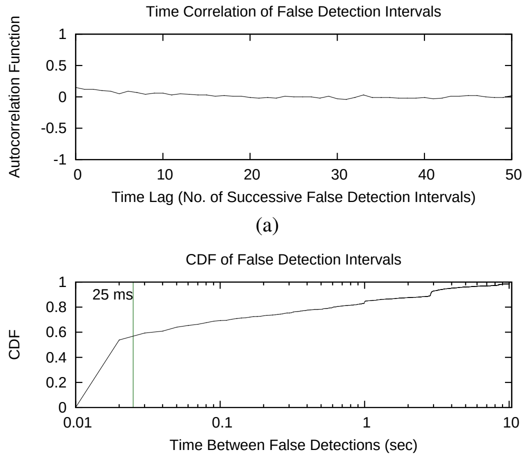
A. *Performance of Link Layer Feedback*

First, we evaluated the PDR and delay of link layer feedback with hello at 25 pkt/sec and as the node failure rate varies from 0 to 4. Although both schemes show significant degradation as node failures increase, hello outperforms link layer feedback by as much as 50% in both performance metrics. This is clearly unexpected since it has been widely reported that link layer feedback can detect link breakages faster and it should therefore have lower packet loss.

To understand the poor performance of link layer feedback, we logged all false detections generated by `ulfra`. We noted that link layer feedback triggers numerous false detections, more than 30 times that of hello when there are node failures. Given that every experiment lasts for 300 seconds, link layer feedback effectively generates an average of more than one false detection per second. This significantly high false detection rate causes severe impact on the operation of the routing protocol. In particular, it causes numerous route discoveries to be invoked, which results in higher routing overload.

B. *Improving Link Layer Feedback*

The results in the preceding section reveals the danger of using *raw* link layer feedback for link failure detection. Note that link layer feedback treats unicast packets as “probes”.



(a)

(b)

Fig. 4. Time correlation and CDF of false detection intervals.

From the experiments, we find that false detections, when taken as a percentage of the total number of unicast packets sent, ranges from 2% to 6%. Although not substantial, the absolute number of false detections becomes significant when the sending rate is high. It is well-known in failure detection theory that systems designed to respond quickly to abrupt changes are necessarily sensitive to certain high frequency effects [22].

To analyze the false detections, we plot the time correlation and cumulative distribution function (CDF) of the false detection intervals, as shown in Figure 4. The auto-correlation suggests the false detections occur at random while the CDF indicates that more than 90% occurs within one second of each other. The fact that a bulk of the false detections (60%) occurs within 25 ms from each other suggests that raw link layer feedback is sensitive to short-term link quality degradations.

To address the above issues, we propose a *veto mechanism*, implemented at the device-dependent module, to minimize the number of false detections generated by device drivers.When a link failure is detected after failing to send a packet, the module starts a link failure notification (`lfn`) timer, instead of immediately informing the device-independent module. When an ACK is subsequently received after successfully sending another packet and the `lfn` timer has not expired yet, a veto action is performed by cancelling the `lfn` timer. When no ACK is received and the timer expires, a notification is sent to the device-independent module.

A critical aspect of the veto mechanism is the selection of an appropriate `lfn` timeout value. We performed experiments and varied the `lfn` timeout period from 0 to 250 ms. From the CDF (see Figure 4b), this range covers more than 70% of the false detection intervals. Figure 5 shows the different performance metrics as the `lfn` timeout period varies from 0 to 250 ms, under two packet sending rates: 25 and 100 pkt/sec. From 0 to 50 ms, the PDR increases more than two-fold at 100 pkt/sec and increases by more than 20% at 25 pkt/sec. There is dramatic improvement in the delay as well, which drops by more than eight times for both packet sending rates.

The improvement in both metrics can be attributed to the significant drop in false detections. From 0 to 50 ms, false detections drop by more than 20 and 8 times at 25 and 100 pkt/sec, respectively. This leads to a significant reduction in routing overhead as AODV performs significantly less route discoveries. The significant performance improvement of link layer feedback arising from the application of a small `lfn` timeout value confirms our hypothesis that short term link quality degradation, in the order of several tens of milliseconds, is responsible for a large number of false detections.

C. Performance Comparison

We now compare the performance of the link layer feedback (with veto mechanism) with hello beaconing. We used a conservative timeout value of 200 ms for the `lfn` timer as it provided stable performance. Note that the timeout poses a slight trade-off. A short timeout ensures rapid failure detection but increases false detections while a long timeout lessens false detections but increases failure detection delay.

In Figure 6, we present the PDR and delay of hello beaconing, link layer feedback, salvaging, and their combinations. ‘Hello+salvaging” denotes the use of hello beaconing in tandem with packet salvaging while “llf+salvaging” represents the use of link layer feedback with packet salvaging.

We studied the impact of node failures and sending rate on the different schemes. The packet sending rate was fixed at 25 pkt/sec for the experiments while the node failure rate was varied (Figures 6a and 6b). Likewise, the node failure rate was fixed at one failure per node per minute for the experiments while the packet sending rate was varied (Figures 6c and 6d).

Impact of Node Failures: The PDR (see Figure 6a) of all the schemes drops almost linearly as the number of node failures increase from 0 to 4. When there is no failure, all schemes deliver around 100% of the packets. When there are node failures, llf and llf+salvaging significantly outperforms both hello and hello+salvaging. Notably, llf delivers 15–20% more packets than hello when there are node failures.

The decrease in the PDR as the failure rate increases is expected. This is due to the fact that when there are more node failures, path switching needs to be done more frequently. A path switch involves three steps: link failure detection, route discovery to find another path to the destination, and re-routing of packets. Since losses due to route discovery and packet re-routing are roughly the same in both hello and llf, the difference in the performance can be attributed to the link failure detection mechanism. It is therefore clear that the rapid failure detection provided by link layer feedback can yield considerable performance improvement.

We now examine the effect of packet salvaging. For hello, salvaging improves its delivery ratio by 8% on the average when there are node failures. Despite this, hello+salvaging still lags behind llf. Salvaging does not give significant performance improvement in llf because the device transmit queues are almost always empty since llf detects link failures rapidly (leaving insufficient time for queues to be filled up.) Thus, only

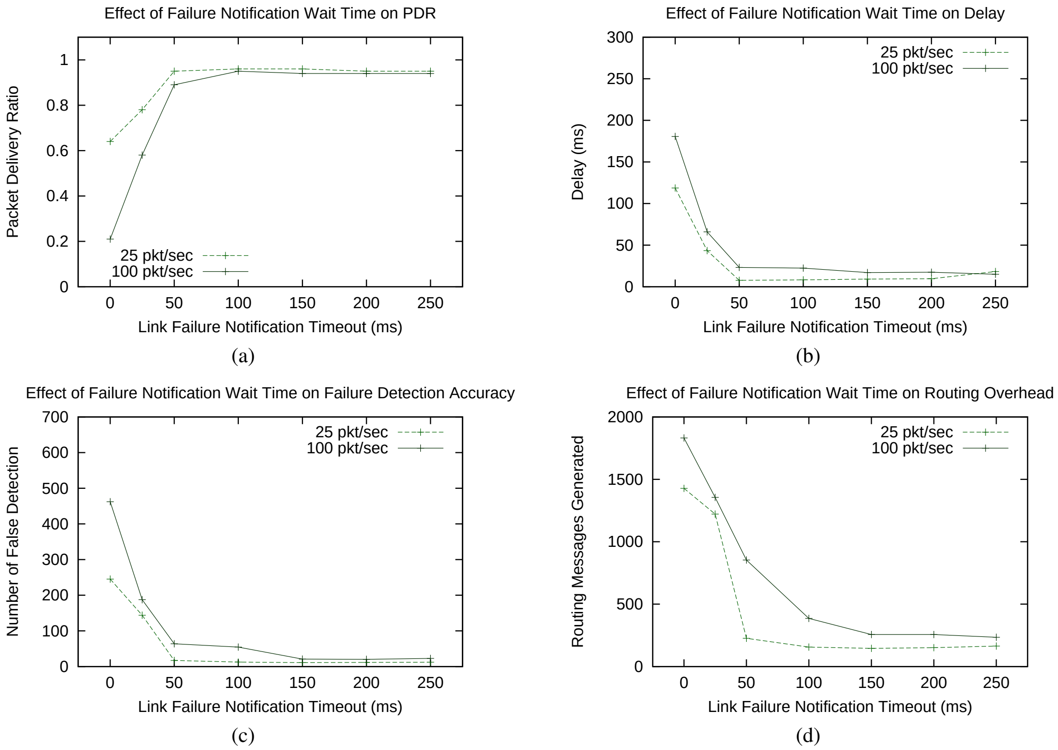


Fig. 5. Performance of link layer feedback with varying lfn timeout values.

a few packets are actually salvaged. In contrast, hello takes a long time to detect link failures and, the transmit queues are filled with substantially more packets. Thus, salvaging causes a significant increase in delay (see Figure 6b) and is clearly not necessary when llf is used. Nevertheless, it can be used when hello is the only available failure detection mechanism.

Impact of Traffic Load: The PDR (Figure 6c) of all schemes drops as the sending rate increases. The significant drop occurs between 100 and 150 pkt/sec. Again, llf and llf+salvaging performs significantly better than hello and hello+salvaging at all sending rates. Regardless of salvaging, llf delivers 12–20% more packets than hello at higher sending rates.

While packet salvaging improves hello and llf when load is constant (6a), the results in this experiment show otherwise. When load increases, salvaging worsens the network contention. Suppose the packet sending rate is R and the link failure detection delay is D_{detn}. Then, the number of packets salvaged during a link failure, N_{salv}, is given by:

$$N_{salv} = R * D_{detn} \tag{1}$$

In addition, AODV also buffers packets while waiting for a route reply. If the router discovery delay is D_{disc}, then the number of packets buffered by AODV, N_{aodv}, is given by:

$$N_{aodv} = R * D_{disc} \tag{2}$$

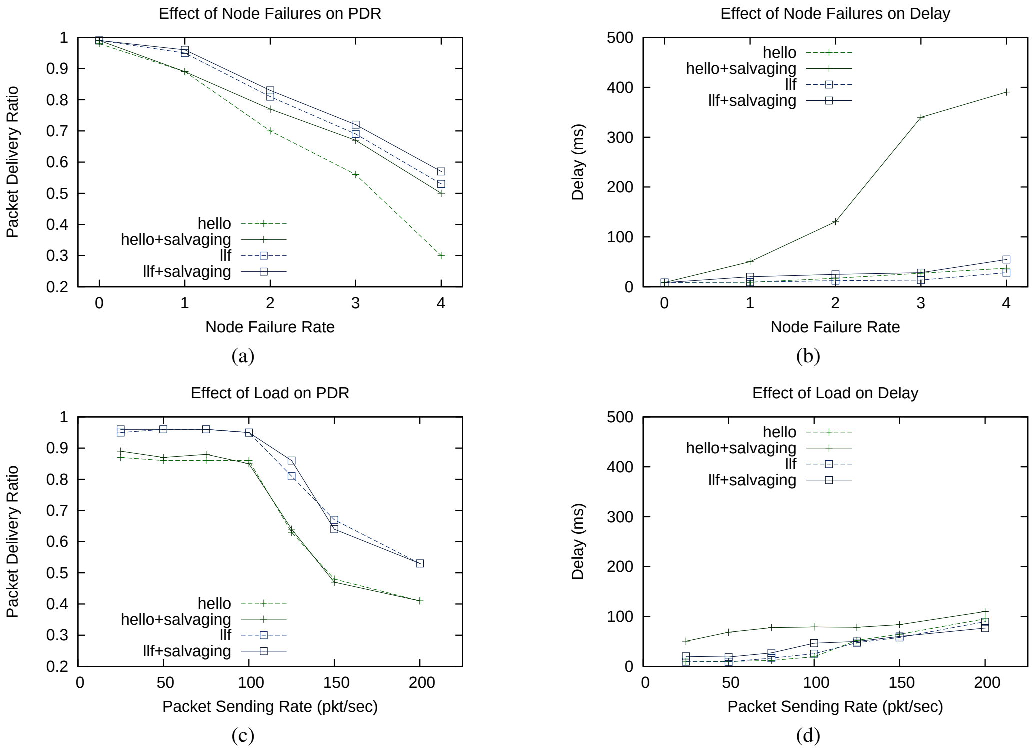
Upon receipt of a route reply, AODV and the salvage cache simultaneously "flush" their packets into the network. The number of flushed packets is $N_{salv} + N_{aodv} = R(D_{detn} + D_{disc})$. If R or $(D_{detn} + D_{disc})$ is large, then the number of flushed packets could be substantial. E.g., let $R = 100$ pkt/sec and $D_{detn} + D_{disc} = 4$ (a value observed when hello is used); then, the number of packets flushed right after route discovery is 400 packets. This would cause significant contention and hence lower the network throughput. One possible solution would be to moderate the flushing of both AODV send buffer and salvage cache after route discovery.

As expected, increasing network traffic consistently increases the delay of all the schemes (see Figure 6d.) At 200 pkt/sec, the delay of all schemes reaches around 100 ms which is significantly higher than their respective delays at 25 pkt/sec. The delay of hello does not significantly increase because salvaging does not contribute to its PDR.

V. CONCLUSION AND FUTURE WORK

In this paper, we proposed a unified link failure detection and recovery architecture (`ulfra`) to improve the resilience of IEEE 802.11 ad hoc networks to persistent link failures. The cross-layer architecture is routing protocol-independent and employs link layer feedback for rapid failure detection and packet salvaging as a recovery mechanism. We implemented and deployed the architecture in a real IEEE 802.11 wireless multihop ad hoc network.

Our experimental study of link layer feedback (as modeled in network simulators) show that it performs worse than hello beaconing. It generates excessive false detections that severely degrades the network performance. The problem is worse at higher network load as the number of false detections is proportional to the packet sending rate. Link layer feedback is



(a) (b) (c) (d)

Fig. 6. Performance comparison of the different mechanisms implemented in `ulfra`.

also sensitive to short-term link quality variations. To address these issues, we incorporated a veto mechanism into the basic link layer feedback to suppress spurious detections. This dramatically improves the performance of link layer feedback in terms of packet delivery, delay, and routing overhead as false detections are considerably reduced. Compared with hello, it delivers 15–20% more packets at high node failure and 12–20% more at high network traffic.

Link layer feedback with the veto mechanism and the `ulfra` architecture clearly needs to be investigated further and, we plan to investigate the effect of mobility, auto-rate, and links with marginal qualities. We also plan to integrate other routing protocols into the architecture, particularly DSR and CHAMP, as they are designed to exploit packet salvaging.

REFERENCES

[1] R. Ramanathan and J. Redi, "A brief overview of ad hoc networks: challenges and directions," *IEEE Communications Magazine*, vol. 40, no. 5, pp. 20–22, May 2002.

[2] D. Aguayo *et al.*, "Link-level measurements from an 802.11b mesh network," in *Proc. ACM SIGCOMM 2004*, Aug. 2004.

[3] O. Bonaventure, C. Filsfils, and P. Francois, "Achieving Sub-50 Milliseconds Recovery Upon BGP Peering Link Failures," *IEEE/ACM Trans. Networking*, vol. 15, no. 5, pp. 1123–1135, Oct. 2007.

[4] I. Chakeres and E. Belding-Royer, "The utility of hello messages for determining link connectivity," in *Proc. IEEE WPMC 2002*, vol. 2, Oct. 2002, pp. 504–508.

[5] Y. Owada *et al.*, "OLSRv2 implementation and performance evaluation with link layer feedback," in *Proc. ACM IWCMC*, 2007, pp. 67–72.

[6] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *Proc. ACM MobiCom 1998*, 1998, pp. 85–97.

[7] H. Wang *et al.*, "Fast neighbor join and link breakage detection for metx-based routing in wireless sensor networks," in *Proc. IEEE Workshop on Embedded Networked Sensors*, 2005, pp. 89–96.

[8] D. Johnson, D. Maltz, and J. Broch, "DSR the dynamic source routing protocol for multihop wireless ad hoc networks," in *Ad hoc Networking*, C. Perkins, Ed. Addison-Wesley, 2001, pp. 139–172.

[9] A. C. Valera *et al.*, "Improving protocol robustness in ad hoc networks through cooperative packet caching and shortest multipath routing," *IEEE Trans on Mobile Computing*, vol. 4, no. 5, pp. 443–457, 2005.

[10] "Network Simulator – ns-2." [Online]. Available: http://nsnam.isi.edu/nsnam/index.php/User_Information

[11] "Qualnet Simulator," Homepage: http://www.scalable-networks.com/.

[12] Y. Huang, S. Bhatti, and D. Parker, "Tuning OLSR," in *Proc. IEEE PIMRC 2006*, sep 2006, pp. 1–5.

[13] C. Perkins, E. Belding-Royer, and S. Das, "Ad hoc on-demand distance vector (AODV) routing," IETF RFC 3561.

[14] B. Bellur and R. Ogier, "A reliable, efficient topology broadcast protocol for dynamic networks," in *Proc. IEEE INFOCOM 1999*, vol. 1, Mar. 1999, pp. 178–186.

[15] R. Gummadi *et al.*, "Understanding and mitigating the impact of RF interference on 802.11 networks," *ACM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 385–396, 2007.

[16] IEEE Computer Society, "ANSI/IEEE std 802.11-1999 (part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications)," Jun. 1999.

[17] E. Nordström, "AODV-UU: Ad-hoc On-demand Distance Vector Routing." [Online]. Available: http://core.it.uu.se/core/index.php/AODV-UU

[18] A. Tonnesen, "olsrd: Ad-hoc wireless mesh routing daemon," Jul. 2009, [Online]. Available: http://www.olsr.org/ [Accessed: Nov 24, 2009].

[19] C. Tschudin *et al.*, "Lessons from experimental manet research," *Ad Hoc Networks*, vol. 3, no. 2, pp. 221–233, Mar. 2005.

[20] V. Kawadia *et al.*, "System services for ad-hoc routing: Architecture, implementation and experiences," in *Proc. ACM MobiSys*, 2003, pp. 99–112.

[21] "The madwifi project," Homepage: http://madwifi-project.org/.

[22] A. Willsky, "A Survey of Design Methods for Failure Detection in Dynamic Systems," *Automatica*, vol. 12, pp. 601–611, 1976.