A General Multiplexed ECN Channel and its use for Wireless Loss Notification

Mayank Sharma, Stanford, msharma@stanford.edu
Dina Katabi, MIT, dk@mit.edu
Rong Pan, Stanford, rong@stanford.edu
Balaji Prabhakar, Stanford, balaji@stanford.edu

***Abstract*— Network-to-transport signaling is desirable for ensuring efficient resource usage and timely notice of network status. ICMP is the standard way for signaling, but unfortunately it generates extra load and does not traverse firewalls. In this paper, we develop M-ECN, an in-band network-to-transport signaling mechanism, which does not generate any extra packets and does not require any dedicated header bits. The key idea is to sneak messages into the stream of ECN bits, *but without* interfering with ECN congestion signaling. Compared to other alternatives, the M-ECN channel is easy-to-deploy because routers read and write to the IP header, and the channel requires no modification to legacy routers along the path which do not participate in the signaling.**

I. Introduction

In this paper, we are interested in network-to-transport signaling, a restricted form of communication in which the network reports to the end-points of a flow the occurrence of a particular event. Signaling is desirable for ensuring efficient resource usage and timely notice of network status. Some example applications are: a router signaling to a sender that a packet loss was due to wireless errors rather than congestion [2], [4], [6], [12], [18], warning a source that it is suspected of misbehaving [19], and informing sources of a drastic change in link capacity [9].

Prior proposals for network-to-transport signaling require either sending out-of-band ICMP messages or using dedicated fields in packet headers. Both approaches have shortcomings: Out-of-band messages generate extra load, do not cross firewalls, and create the potential for misuse (e.g., an attacker may misinform a victim that the path capacity has been drastically reduced). Using dedicated fields in the IP header is problematic given that header real-estate has already been allocated to different functionalities. The use of additional application-dependent headers requires a change to router architecture, interferes with tunnels, and violates end-to-end semantics.

We propose multi-functional header fields which can simultaneously be used by more than one task. Specifically, we develop the Multiplexed ECN (M-ECN) channel as a means for allowing routers and end-systems to communicate low rate information using ECN, but without interfering with ECN's signaling of congestion. The key observation we leverage is this: The stream of ECN bits from a flow creates a communication channel whose capacity can be up to one bit per packet. But because the ECN mark is infrequently set, the rate of information transfer over this channel is much smaller than its capacity. Thus, there is some left-over capacity to communicate other information. To use this spare capacity, the router needs to spread a message across multiple packets of a flow, while ensuring that ECN congestion marks are given priority over other information.

Our contribution is twofold. First, we introduce the M-ECN channel as a way of designing multi-functional header fields. Prior work on using a field in the IP header for a new task (which differs from the field's actual purpose) did not explicitly require the (peaceful) co-existence of different functions on a single header field [27], [28]. Modeling the M-ECN channel as a "**Z**-channel with erasures" allows us to information-theoretically evaluate the capacity of the M-ECN channel and compute the spare capacity available for other information. Some other features of M-ECN are:

1. It is amenable to gradual deployment. Only routers and end-points which participate in communication over the M-ECN channel need to be modified. The channel is transparent to legacy routers.
2. It can be simultaneously used by multiple routers along a path to communicate with the end-points of a flow.
3. Different flows can use it for different applications.
4. It uses ECN-Nonce to prevent a receiver from conveying the wrong M-ECN signal to the sender.
5. It is resilient to packet drops and reordering.

Second, as a concrete application of the M-ECN channel abstraction, we develop and evaluate a novel protocol called WiSE, for signaling wireless error drops to the source. Such signaling is known to significantly improve TCP performance over wireless links. We devise a simple scheme for signaling packet corruption on the M-ECN channel via ECN mark positions. We show through simulation that WiSE achieves a performance gain comparable to previous end-to-end wireless error recovery mechanisms, such as ELN [2] and ETEN [18], *but without* generating extra messages or requiring dedicated bits in packet header. Further, WiSE is fair to non-WiSE sources and reacts properly to ECN congestion marks.

This paper is organized as follows. In §II we motivate our work by mentioning the limitations of signaling mechanisms that have been proposed in the literature. Section III provides some background on ECN. The M-ECN channel abstraction is introduced in §IV, where we also define the concept of *opportunistic signaling* and calculate the capacity of the M-ECN channel. We propose WiSE, a mechanism for wireless loss notification that uses M-ECN in §V. Section VI provides a detailed specification and performance evaluation (§VI-C) of the WiSE protocol. In §VII we mention some salient features and benefits of M-ECN (and thus WiSE). As a proof of concept of another application of M-ECN, in §VIII we propose a mechanism called XECN, which helps large flows traversing links with high capacity to regulate their window size in response to availability of bandwidth. Finally we conclude in §IX.

II. Limitations of Prior Signaling Mechanisms

Current mechanisms for network-to-end-points signaling are sometimes inadequate or suboptimal, as argued below.

(A) Out-of-Band ICMP Messages: ICMP is the standard way for communicating information between network and users. Unfortunately, as a signaling mechanism, ICMP has many limitations. First, ICMP messages are filtered out by most firewalls because they create a security hazard and a potential for a denial-of-service attack [5]. Second, ICMP messages create extra load and hence should not be used too often. Third, ICMP does not deal with IPsec [17] or tunnels. Fourth, ICMP may be inappropriate for certain applications. For example, consider using ICMP messages to signal error drops over a multi-hop wireless network. When fading occurs the quality of the wireless channel degrades causing a large number of packet losses in a short interval. The router will generate ICMP messages to signal each corrupted packet to its sender. But, in a multi-hop wireless network, the router will need to use the same erroneous wireless channel to send the ICMP messages. So the ICMP messages will compete with the data packets for the low quality wireless channel causing even more errors and drops, and consequently more ICMP messages.

(D) IP options: Information may be exchanged between routers and end-systems in IP options. Currently the use of IP options in packets removes them from the fast path, and causes them to incur substantial delays. Thus, IP options are not desirable for signaling repeated events (e.g., wireless error drops). In general, the problem with IP options is gradual deployment; even if one changes the signaling router to use IPv6 or to implement IP options efficiently, legacy routers along the path will either drop the packet (if it is IPv6) or send it along the slow path.

(B) Dedicated Bits in the IP or TCP Header: Using dedicated fields in the IP header is problematic given that header real-estate has already been allocated to different functionalities. Allowing routers to write to fields in the TCP header requires the router to recompute the TCP checksum for each packet, violates end-to-end semantics, interferes with tunnels, and makes it hard to use the same service with other transport protocols.

(C) Additional Dedicated Headers: Conceptually, it is possible to signal using an additional service-dependent header that intervenes between the TCP and IP headers. In practice, this causes deployment problems, might require changing router architecture, and each new application will define a new header. Currently routers strip the first few bytes of a packet which usually contain the IP and TCP headers. Inserting additional headers between IP and TCP may prevent access to the TCP header, and confuse legacy routers which want to read some TCP fields (e.g. port numbers) but access the new header instead.

ECT CE	Meaning
00	ECN incapable
10	no congestion, Nonce=0
01	no congestion, Nonce=1
11	congestion, Nonce cleared

(a) ECN encoding in the IP header (see §VII-A for nonce definition)

ECN-Echo: Set by the receiver in TCP ACK to signal to the sender the reception of an ECN marked packet.
CWR: Set by the sender to acknowledge its receipt of and reaction to the ECN-Echo flag.
NS: The receiver returns the cumulative sum of the nonce in the NS bit in the TCP ACK.

(b) ECN bits in the TCP header

Fig. 1. The bits used by ECN in both the IP and TCP headers.

III. Background & Terminology

(A) Explicit Congestion Notification (ECN): ECN allows routers that use active queue management (AQM) to mark packets as an indication of imminent congestion. This has the potential of limiting packet drops due to a buffer overflow. ECN uses the ECT and CE bits in the IP header for signaling between routers and connection endpoints, and uses the ECN-Echo, CWR, and NS bits in the TCP header for signaling between sender and receiver. Figure 1 defines the meaning of these bits as presented in RFC 3168 [25].

(B) Terminology: For simplicity, we will always refer to ECN as being one bit. Thus, ECN = 1 means that the CE and ECT flags are "11", whereas ECN = 0 means that the flags are either "01" or "10".

IV. The M-ECN channel

M-ECN is an in-band easy-to-deploy mechanism for network-to-transport signaling. We envision that some routers in the network may be equipped to signal certain events. For example, a subset of the wireless routers may be able of signaling packet corruption; some other routers may be equipped to signal quality of service, etc. Each one of these services will have a unique well-known identifier. Using its first packet, a flow (identified by its sender, receiver, and ports) may subscribe to a particular M-ECN signal by specifying the service identifier in an IP router-alert option [15]. (Subsequent packets of the flow do not use IP options).

Routers signal to the end point of a flow using messages constructed out of ECN marks without interfering with ECN congestion signaling (see Figure 2). The key idea is to use *opportunistic signaling*; i.e. to sneak messages into seldom-used standardized fields, while giving priority to the original functionality of the field. In §V-C, we discuss in detail how to construct M-ECN messages, but at a high-level opportunistic signaling works as follows:

1. It spreads an M-ECN message across multiple packets. We call the sequence of ECN values (i.e. string of 1's and 0's) in a single M-ECN message a **Codeword**. Spreading a codeword across multiple packets requires a router to maintain

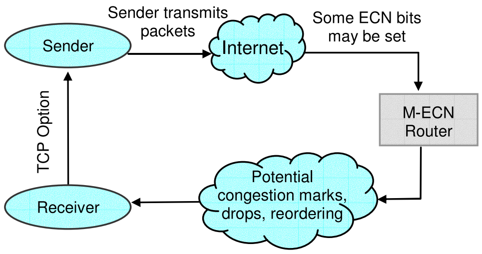

Fig. 2. M-ECN routers signal to the receiver using code-words constructed out of ECN marks. If needed, the receiver can relay the message to the sender using an end-to-end mechanism such as a TCP option.

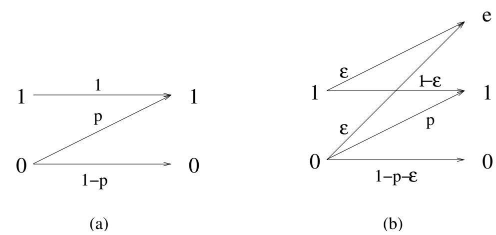

Fig. 3. (a) The **Z**-channel (b) The **ZwE**-channel

per-subscriber state. Although this limits the number of flows to which the router can signal at high speed, the space of possible applications is still substantial. For example, both wireless loss notification and warning a suspected misbehaver are potential applications, since in the former the bottleneck is the link bandwidth not the router, whereas in the latter the router would need to maintain state to detect misbehaving regardless of whether it uses the M-ECN channel [19].

2. It encodes information in patterns (i.e., strings) of 0's & 1's that are unlikely to be generated by ECN congestion marking.

3. It gives priority to ECN congestion signaling over sneaked messages. First, whenever an AQM router along the path marks a packet because of congestion, it results in an illegal M-ECN codeword which allows the end system to recognize the congestion mark. Second, M-ECN codewords have a very low density of 1's, leaving most of the packets available for congestion marking by AQM routers.

4. To construct M-ECN codewords, we use the observations: 1) ECN marks can be swapped between two packets that are closely spaced in time; 2) a cluster of ECN marks is equivalent to a single mark since TCP reacts to only one decrease signal in a congestion window (`cwnd`). Similarly, an ECN mark inserted after a drop is harmless because TCP reacts only to one congestion signal per cwnd.

We believe that the concept of opportunistic signaling may be applicable to fields other than ECN (e.g. the DiffServ bits). We have chosen to focus on ECN because it is standardized, has been widely deployed in both routers and end systems, and being in the IP header, is easily accessible to routers for reading and writing. Further, there are a few proposals for using ECN to signal alternative information such as congestion price [13]. These proposals usually assume that all routers along the path have been updated to understand their alternative meaning of ECN. But if some routers perform conventional ECN congestion marking and others use the alternative meaning, the resulting signal will contain unpredictable errors. Hence, some of these proposals may be able to use M-ECN for gradual deployment to allow them to deliver a signal even when the path traverses legacy ECN routers.

A. The Capacity of the M-ECN channel

Using methods from information theory, we model the M-ECN channel as a binary channel with erasures and compute the left-over capacity after congestion marking. In the canonical formulation of the communication problem, there is a transmitter communicating to a receiver over a noisy channel. Here, the transmitter is an M-ECN router and the receiver is a flow's end-point. The effect of ECN congestion marking on M-ECN messages is modeled as channel noise. This noise flips a 0 to 1 with the congestion marking probability p, but it faithfully transmits a 1 with probability 1, as illustrated in Figure 3(a). The resulting channel is the **Z**-channel of information theory. The capacity of the **Z**-channel is known [14], and asymmetric error correcting codes have been constructed and shown to perform well [24].

While the **Z**-channel models the congestion marking process at a regular AQM router, it does not model the effect of packet drops. Information theoretically, packet drops lead to the "erasure" of a 0 or a 1, which we model by the **Z**-channel with erasures (or the **ZwE**-channel), shown in Figure 3(b). In the **ZwE**-channel a 0 or a 1 is erased with probability ϵ, which is the drop probability along the path. When an erasure (i.e. a packet drop) occurs, we receive an e (i.e. a gap in the sequence number) instead of a 0 or a 1.

We shall now compute the capacity, $C(p, \epsilon)$, of the **ZwE**-channel as a function of the congestion marking probability p and the drop probability ϵ. This allows us to understand how much spare capacity remains for transmitting extra information over the M-ECN channel. By definition, (see [7]) the capacity of a discrete memoryless channel is:

$$C = \max_{0\leq q\leq 1} I(X;Y) = \max_{0\leq q\leq 1} H(Y) - H(Y|X), \quad (1)$$

where $I(X;Y)$ is the Mutual Information between the input X and the output Y, $H(Y)$ is the entropy of the output, $H(Y|X)$ is the conditional entropy of the output given the input and the maximum is over all possible probability distributions on the input. For the **ZwE**-channel, the inputs X are 0 or 1, and the possible outputs Y are 0, 1 or e. We want to express this capacity as a function of the density of 1's in the code q because we want to ensure that M-ECN messages have a low density of ones so that there are enough unmarked packets to allow a regular AQM router to signal congestion through marking. Thus,

$$\begin{aligned} I(X;Y) = H\left(\epsilon, q(1-\epsilon) + (1-q)p, (1-q)(1-p-\epsilon)\right) \\ -qH\left(\epsilon, 1-\epsilon\right) - (1-q)H\left(\epsilon, p, 1-p-\epsilon\right) \quad (2) \end{aligned}$$

and the capacity $C = C(p, \epsilon)$ can be computed as an explicit function of p, ϵ, and q (details of the derivation are in the appendix).

To give a feeling for the result, we plot in Figure 4 the capacity for the particular case where $p = 0.01$ and $\epsilon = 0.01$. The figure shows that there is reasonable capacity left even if

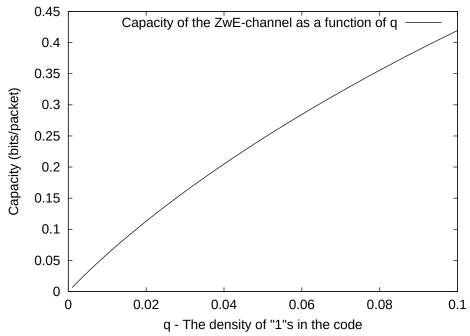

Fig. 4. Capacity of the **ZwE**-channel as a function of the density of 1's for congestion marking probability $p = 0.01$ and drop probability $\epsilon = 0.01$.

we constrain our codewords to have a very low density of 1's. For example, if the congestion marking probability $p = 0.01$, the drop probability $\epsilon = 0.01$, and one allows the density of ones in the codewords to be 0.02, then the M-ECN channel provides $0.11-p = 0.1$ bits/packet to signal extra information. Said differently, on an average, if we had a sequence of 100 packets of which 1 got randomly dropped and 1 randomly marked, and we were allowed to use only 2 ones (i.e., 2 marks) to communicate a message, then we could transmit up to 10 bits of information using this packet sequence.

To construct M-ECN messages, one needs to follow the rules of opportunistic signaling in §IV. Next, we consider a particular application called WiSE and in that context, devise a simple scheme which constructs messages using the position of ECN marks relative to wireless drops.

V. WiSE: Wireless Signaling via ECN

As a concrete application of M-ECN, we propose WiSE, a protocol for signaling wireless losses to a TCP sender over the M-ECN channel. WiSE enables routers to encode information regarding the cause of each of a flow's packet-drops in the stream of ECN bits belonging to that flow. Using this information, the sender can react more intelligently to packet losses, thus improving the throughput of the network.

A. Problem & Previous Solutions

The wireless channel is well-known for exhibiting error rates substantially higher than those observed in wired environments. Although current access technologies have mitigated this problem to some extent for the last hop, short range, indoor environment, they are still of concern in designing high-rate channels for outdoor or mobile environments [8], [16].

TCP reacts adversely to the high error rate seen on wireless links because it interprets any packet drop as a sign of congestion. Thus, TCP reacts to wireless packet losses by unnecessarily halving its congestion window.

Various strategies have been suggested in the literature to combat this problem [1], [2], [3], [4], [6], [12], [18], [20], and they can be classified into two main classes. The first class shields the sender from the wireless link by using local link-layer retransmissions. The second class, to which WiSE belongs, exposes the reasons for the drops to the end-user thus allowing the sender to recover appropriately.

WiSE Contributions: In addition to serving as a case study of the M-ECN channel, WiSE provides a performance gain higher than ETEN [18] and comparable to ELN [2], without using additional message packets or changing the header structure. In contrast to the current version of ELN, WiSE does not assume route symmetry and works even if a flow traverses multiple wireless hops. Finally, WiSE signaling is simple and can be implemented at a router using 3 lines of `C` code.

B. Overview of the WiSE protocol

The WiSE protocol is implemented through two components: the WiSE-Agent and WiSE-TCP, as shown in Figure 5. Located at the router, the WiSE-Agent is a data-link layer module that detects wireless packet errors and signals them to WiSE-TCP over the M-ECN channel. WiSE-TCP is an extension to standard TCP which allows it to decode the "packet-drop type" information sent by the WiSE-Agents along a flow's path. We assume 802.11, where a corrupted packet is discarded at the sending MAC after zero or more retrials. Thus, before dropping a packet because of wireless errors, the router can read the ECN value in the packet.

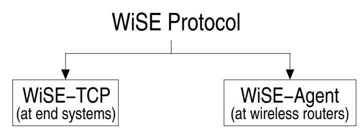

Fig. 5. The WiSE Components

Below is an overview of how WiSE works (the implementation details are in §VI.

1. The first packet subscribes the flow to the WiSE service using a router alert IP option [15].

2. Whenever 802.11 discards a packet because of a wireless error, the WiSE agent signals this information in a message sent over the M-ECN channel (as described in §V-C). The agent preserves ECN marks from upstream.

3. Receiver decodes the WiSE signal and interprets illegal codewords as congestion. It informs the sender about error losses using a TCP option. (Routers do not access this option).

4. Sender recovers from errors without reducing its rate.

C. Legal & Illegal Codewords

Our objective is to make corruption drops look very different from congestion drops by associating them with a very structured pattern of ECN bits that is unlikely to be created randomly. Whenever a WiSE router (i.e. agent) drops a packet because of corruption, it inserts a 1 in the stream of ECN bits from the flow. Further, the WiSE router preserves the 1's in the ECN field of packets coming from upstream. Thus, if a WiSE router drops a marked packet because of wireless errors, then, in addition to inserting a new mark, the router remembers the ECN mark on the dropped packet and moves it to a later packet from the same flow. The same happens if the router sets a WiSE mark on an already marked packet.

The resulting codewords used by WiSE start with the occurrence of a drop and end with the occurrence of ECN=0, and are characterized by having an equal number of drops and ECN marks. For example, let X represents a drop then X10, X1X10, and XX110 are all legal WiSE codewords. Further, at any point through a legal codeword, the number of drops equals or exceeds the number of ECN marks; e.g. X11X0 is not a legal codeword. Any sequence of drops/marks that doesn't satisfy the above properties is a sign of congestion.

D. Example Scenarios

In the examples, we will only show a flow's ECN bits and represent them as a stream. The ECN field of a dropped packet will be marked with an X. Each packet, and thus the ECN bit associated with it, will be referenced by its *sequence number*.

Example 1: Successive drops at a wireless link
Consider the situation in Figure 6, where packets with sequence numbers 5 and 6 are both dropped because of wireless errors. Note that WiSE will set to 1 the ECN fields of the next two successfully transmitted packets belonging to that flow. The receiver will get the valid WiSE codeword XX110, and convey to the sender that packets 5 and 6 have been discarded because of errors.

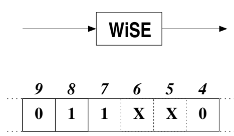

Fig. 6. Successive error-caused drops at a wireless link

Example 2: Composability of WiSE Messages
This example (Figure 7) shows that WiSE encoding is accumulative in nature, and thus multiple routers in an ad-hoc network can compose their signals. Packet 5 is corrupted at router WiSE-1 leading to the ECN mark of packet 6 being set to 1. But, packet 6 is corrupted at WiSE-2. So, WiSE-2 has to set ECN=1 on the next two packets to account for the corruption of packet 6 and the fact that packet 6 was already marked. The receiver gets the valid WiSE word XX110, indicating that packets 5 and 6 were dropped because of errors.

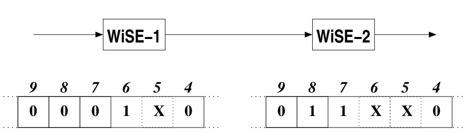

Fig. 7. Composability of WiSE messages

Example 3: WiSE interaction with AQM routers
This example demonstrates that WiSE signaling does not hamper the congestion signals sent by AQM routers. As in Figure 8, the AQM router marks packet 6 with a 1 to signal congestion. But packet 6 is discarded at the wireless link because of errors. Then the WiSE router not only marks a subsequent packet (packet 7) to signal a wireless drop, but also reads the "ECN" mark on the dropped packet and preserves it by setting the ECN bit on packet 8 to 1. The receiver will recognize X110 as an illegal WiSE codeword (since, marks $\neq$ drops) and convey the congestion signal to the sender.

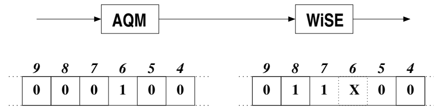

Fig. 8. WiSE interaction with an AQM router

VI. Specifications of the WiSE Protocol

The WiSE protocol is implemented through two components: the WiSE-Agent and the WiSE-TCP modules.

A. The WiSE-Agent

The WiSE-Agent is a data link layer module that detects packet errors over the wireless channel and signals them to WiSE-TCP over the M-ECN channel. The objective of the WiSE-Agent is to signal to the sender that a particular packet was dropped due to corruption on the wireless channel rather than due to congestion. WiSE identifies a flow by its sender and receiver IP addresses and the sender and receiver ports. The WiSE-Agent maintains a list of all flows traversing the wireless link. This does not add any extra overhead to most current base stations since they already collect per-flow state. In ad hoc mode, the overhead is acceptable since wireless routers are limited by the wireless link's bandwidth rather than memory. The WiSE-Agent associates with each flow in the list a field called WiSE-Marks, which counts the number of pending WiSE marks that should be inserted in subsequent packets. The size of this field defines the maximum burst of errors a WiSE-Agent can signal to the sender. For example, if the WiSE-Marks field is chosen to be 3 bits then, the WiSE-Agent can signal a maximum error burst of 8 packets.[1]

The code-words that the WiSE-Agent uses to signal a packet loss are quite simple. When a packet is dropped due to wireless errors, the WiSE-Agent sets the ECN bit on the next packet from the same flow with an unmarked ECN bit. In Section VI-B we show how a WiSE TCP sender decodes this signal. Below, we explain the details of how the WiSE Agent detects a packet loss, finds out which flow the packet belongs to, and encodes the WiSE signal.

Detecting a wireless packet loss:
Located at the data link layer, the WiSE-Agent senses arrivals and departures on the wireless link. It detects a packet loss due to wireless errors in the following two situations:

(a) This case represents 802.11 and is what we expect to see in practice. The data link layer at the sending side aborts transmission over the wireless link after exhausting all of the allowed number of retrials. The WiSE-Agent at the sending side detects this drop. The packet IP header is uncorrupted and the WiSE-Agent can unambiguously determine which flow the packet belongs to, and the ECN bit on this lost packet. The agent uses this information in signaling the loss as described below.
(b) A corrupted packet is received on the wireless link and the sender is not expected to repeat the transmission.

[1] An 8-bit field seems quite reasonable given the statistics of packet errors reported for a wide range of wireless environments [11], [10], [26].

This might happen in two cases. First, the data link layer mechanism does not expect the receiver to ACK a correctly received packet (e.g., a strong FEC code is used). Second, the packet is received correctly from the perspective of the data link layer but does not pass the IP checksum, which the WiSE-Agent has to perform on received packets. When a corrupted packet is received, the source and destination addresses, the source port, and the ECN bit in the header might be corrupted. However with a high probability, the WiSE-Agent can still detect the identity of the flow by mapping the sender-destination IP addresses and the source port to the closest entry in its flow list. As for the ECN bit, the probability that the ECN bit is corrupted out of all the bits in the packet is very low and statistically insignificant.

Signaling a wireless packet loss:
As mentioned above, with each flow, the WiSE-Agent associates a particular field, called WiSE-Marks, that helps it to determine whether there is a pending corruption that should be signaled to the source over its ECN channel. The WiSE-Marks field is initialized to zero which means no pending corrupted packets. The count of packets belonging to a flow that were corrupted at a router due to wireless channel errors is encoded in the number of ECN bits marked. The WiSE algorithm requires an encoding routine at each WiSE-Agent, and a decoding routine at the sender end. Pseudo-code for both is presented below (the decoder's description and pseudo-code is in the next section).

When a corrupted packet is discarded, the WiSE-Agent increments `WiSE-Marks[flowID]` by 1 plus the ECN mark. It waits for an uncorrupted packet, checks if its ECN bit is marked, and if not, sets the bit to 1, as shown below.

```
if (Packet_Drop_Because_of_Corruption)
    WiSE-Marks[flowID] += 1 + ECN_bit;
else if (!Packet_Marked)
    if (WiSE-Marks[flowID]-- > 0)
        ECN_bit = 1;
```

B. The WiSE-TCP

WiSE-TCP is an ECN-enabled TCP sender that can decode WiSE signals sent over the M-ECN channel. In our current implementation of WiSE-TCP all of the WiSE functionalities are kept at the sender. The receiver conveys the stream of ECN values to the sender in a TCP option. We need to convey only one bit in each Ack. we call that bit "Echo". (We need to use our own "Echo" bit which is different from the ECN-Echo bit in the TCP header because ECN is sticky; i.e., when it sees an ECN mark, the receiver is supposed to set the ECN-Echo in all Acks until the sender reset the CWR flag.) A WiSE-TCP sender is different from standard TCP only in the addition of a decoder that is consulted by TCP's congestion control module before reacting to any drop that it discovers. The decoder simply parses the sequence of ECN marks in the TCP option in the ACKs that the sender receives, identifies WiSE marks, and disables the cutting of window size whenever it thinks that a channel error has been signaled to it as the cause of a drop.

The WiSE-Decoder: The WiSE-Decoder receives all acknowledgments (ACKs) and passes them to the congestion controller after decoding the information embedded in the WiSE TCP option bit. By decoding we mean tracking all the ones and identifying those that were marked by WiSE-Agents (to signal drops caused by channel errors) along the path of a flow. These WiSE marks are detected when there is a successful match between a drop discovered at the sender and an ECN mark on one of the ACK packets received in a certain window around the drop. The decoding algorithm performs the above matches by taking into account properties of the WiSE codewords and congestion related ECN marks.

Recall that a WiSE codeword starts with the occurrence of a drop (i.e., a hole in the sequence numbers) and ends with an unmarked packet (ECN = 0), with the number of marks being equal to the number corruption drops.[2] Thus, if TCP receiver would give the sender an accurate report of the holes in the sequence numbers of the packets received, such as in TCP SACK [21], detecting drops and matching them with the corresponding WiSE marks is relatively easy. In this case, the decoder at the sender can detect the beginning of a codeword and match each drop with a subsequent ECN mark, until the arrival of an ACK which has Echo = 0. But in most TCP implementations, the receiver sends only accumulative ACKs and no further information about how many packets are losts. The repetition of a dup-ACK (for duplicate ACK) is a good indication of the occurrence of a drop, and standard TCP uses 3 dup-ACKs as a sign that the packet with sequence number equal to the last ACKed sequence number plus one has been dropped. However, when multiple drops occur in the same congestion window, it could take a little while to discover the second or subsequent drops from amongst them by using repeated dup-ACKs (or eventually a timeout). The reason for this delay is that the receiver has to recover from the first packet drop before it can start sending dup-ACKs for the second drop in the same congestion window. As a result, the ECN bit set by a WiSE-Agent declaring the second drop in the window to be an error, might arrive an RTT (round-trip time) earlier than the actual discovery of the corresponding drop by TCP.

Since this delay in discovering multiple drops in a congestion window is the source of most of the complexity of our otherwise simple decoding algorithm, we provide an example to better explain the problem. Assume that `cwnd`= 6 packets, and that the packets with sequence numbers 10 to 15 are outstanding. Suppose also that packets 10 and 13 are corrupted by wireless errors. Table I shows the ACK sequence numbers and their corresponding Echo bit in the first column. Since packet 10 is lost on the wireless channel, the WiSE-Agent discovers the drop at the router and marks the ECN bit on packet 11. Suppose packet 11 arrives at the receiver without

[2]The drops in a valid codeword are corruption drops. The occurrence of a congestion drop in a codeword destroys the balance between the number of drops and that of marks and renders the codeword invalid.

ACK seqno	Echo	seqno at rx	Remarks
9	0	9	
9	1 (10 corrupted)	11	direct Match
9	0	12	retransmit 10
9	1 (13 corrupted)	14	mismatched WiSE mark
9	0	15	
:	:	:	
12	0	10	retransmission of 10 was successful

TABLE I

EXAMPLE OF THE ACKS ARRIVING AT A RENO SENDER AND THEIR CORRESPONDING ECHO BITS. IN THIS CASE, PACKETS 10 AND 13 ARE LOST ON THE WIRELESS LINK. THE WISE MARK FOR PACKET 10 ARRIVES WITH THE FIRST DUP-ACK FOR PACKET 9. IN CONTRAST, THE WISE MARK FOR PACKET 13 ARRIVES ON A DUP-ACK FOR PACKET 9, BEFORE THE SENDER DISCOVERS THE LOSS OF PACKET 13.

any error; it generates a dup-ACK for packet 9. The receiver then copies the WiSE mark from the ECN bit in packet 11 to the Echo bit in the TCP option in this ACK.

When the sender receives the first dup-ACK for packet 9 with the Echo bit set, the WiSE-Decoder is able to match the discovered drop with a WiSE mark. So it declares 10 as a possibly corrupted packet, but takes no specific congestion action. Packet 10 is declared as a "direct match" since the ECN bit arrived on the first dup-ACK. After 3 dup-ACKs for packet 9 are received, TCP's congestion control module concludes that packet 10 was dropped, and also learns by consulting the WiSE-Decoder that packet 10 is likely to be a corrupted packet. Thus, it retransmits packet 10 without halving the congestion window. Next let us consider instead, a problematic situation, that is, what happens to packet 13 which is also corrupted by the wireless link. Packet 14 is WiSE marked, and the resulting ECN mark is copied at the receiver to one of the dup-ACKs for packet 9 (See Table I). Note that packet 9 is still being dup-ACKed because packet 10 has not yet been successfully received. As such, the WiSE mark arrives at the sender before the sender detects the loss of packet 13 (mismatched WiSE mark in the Table). This asynchrony is a result of the accumulative nature of TCP ACKs and does not occur when TCP uses the SACK option.

Despite the problematic nature of accumulative ACKs, we can still decode the WiSE signal by having our decoder remember ECN marks that have arrived while a sequence number is repeating, and by conducting a more sophisticated test for matches between drops discovered by TCP and WiSE marks, than the direct match that happened to work for packet 10 above. These pending ECN marks will be stored until the sequence numbers start advancing normally (i.e. incrementing by 1 per ACK, or 2 per ACK in case of delayed ACKs).[3]

We now formally describe the decoder. The first component of the decoder is the module that identifies candidates for drops that some WiSE-Agent might be signaling as being a wireless error. A packet is identified as a potential drop when the first dup-ACK for it is received. This is only a heuristic to identify potential candidates for drops that TCP congestion control will discover later (after 3 dup-ACKs or a timeout). But this should not be misunderstood to mean that the decoder "discovers" drops on its own from the first dup-ACK. Throughout we also maintain a counter `Packet_Position`, which tracks the number of dup-ACKs received at the sender since the last time the sequence number increased. In particular, `Packet_Position` is set to one each time we see the first dup-ACK for some sequence number.

[3] Actually, the sequence number should increment twice before we can tell that the sequence numbers are advancing normally. This takes care of situations in which both packets x and $x + 1$ were dropped.

DROP IDENTIFICATION MODULE

```
if (First Repeat_Ack) {
    Packet_Position = 1;
    New_Packet_Drop = 1;
}else {
    New_Packet_Drop = 0;
    Packet_Position++;
}
```

The decoder maintains the following variables: a queue `Pending_Marks` of tuples with four fields (seq, pos, cwnd, match) that stores the sequence number, the `Packet_Position`, the `cwnd`, and a flag corresponding to each unresolved ECN mark at the time of its arrival at the sender; a flag called `New_Packet_Drop` which is set on receiving the first dup-ACK for a sequence number. TCP uses the queue to resolve the state of a drop using the Match function described below. If there is a match, TCP retransmits the dropped packet but doesn't reduce `cwnd`). The pseudo-code for the decoder is as follows:

DECODER

```
if (Sequence Number Advancing Normally) {
   if (ECN_bit) {
      Perform ECN action;
      Empty Pending_Marks;
   }
} else {
   if (ECN_bit) {
      i = New tuple;
      i.pos = Packet_Position;
      i.seq = seq in ACK;
      i.cwnd = cwnd;
      if (New_Packet_Drop)
         i.match = 1;
      Pending_Marks.enque(i); //At tail
   }
}
```

The expression "Sequence Number Advancing Normally" means that both of the most recent ACKs incremented the sequence number (This takes care of situations in which both packets x and $x+1$ were dropped.) The above code shows that as long as sequence numbers are advancing normally we react to ECN congestion marks. When the sequence number does not advance we consider this as an indication of a potential drop and we start collecting the ECN Marks.

When the TCP congestion control code discovers a drop, it calls the Match function which return a zero if the drop is a congestion drop and a 1 if the drop is a corrupted packet. The function `Match()` is implemented as below:

```
function Match (int seq) {
   if (Pending_Marks) {
      tuple * p = Pending_Marks->head;
      if (p->match)
         return 1; // Direct Match
      else
         if (p->pos+p->cwnd >= seq)
            return 1;
         else{
            Pending_Marks->deque(p);
            return 0;
         }
   } else return 0;
}
```

The Match function takes the sequence number at the time of the discovery of the drop (`seq`) and matches it against the entries in the `Pending_Marks` queue. (Note that Match() is not guaranteed to discover all corruptions. In particular, the Echo bits could be dropped on the reverse path which prevents a matching of the number of data packet drops in a valid codeword with the corresponding WiSE marks.)

C. *WiSE Evaluation*

(A) Simulation Environment: Our simulations use *ns-2* [23], which we have extended to include a WiSE-Agent (derived from RED queue) and a WiSE-TCP (a modified NewReno) module. (We have also compared WiSE-SACK with standard SACK, which lead to results similar in nature to those below.) For wireless errors, we use a two-state Markov error model. As per data cited in studies of the wireless channel [22], we set the average burst size of errors in the Markov model to 3 packets. Finally, in all simulations, wired routers use RED queues (max= 2×min = 1/3 buffer), the buffer size is a bandwidth-delay product, packets are 576 bytes, the round trip time (RTT) is 50 ms, and the ECN option is on.

(B) WiSE boosts performance over wireless links: In this experiment, we have a 40 Mb/s wireless link traversed by 700 web sessions with Pareto distributed file sizes having an average of 12 pkts/file; 10 long-lived FTPs and reverse traffic. Average packet error rate is 0.01. Figure 9 shows the boost in total throughput due to WiSE when the traffic mix is similar to that expected in the Internet.

(C) Robustness & TCP-Friendliness: In this experiment, the bottleneck is *wired*. Two flows share a congested 2 Mb/s RED+ECN router. One of these flows also traverses a 10 Mb/s wireless link with error rate 0.01. Figure 10(top) shows that when both flows are NewReno the wireless flow suffers. In contrast, when the wireless flow uses WiSE, as in Figure 10 (bottom), its performance significantly improves. The experiment reveals: 1) WiSE works correctly when there are non-WiSE routers along the path; 2) The fact that WiSE is fair to the non-WiSE flow shows that WiSE reacts properly to ECN marks from the congested RED+ECN router.

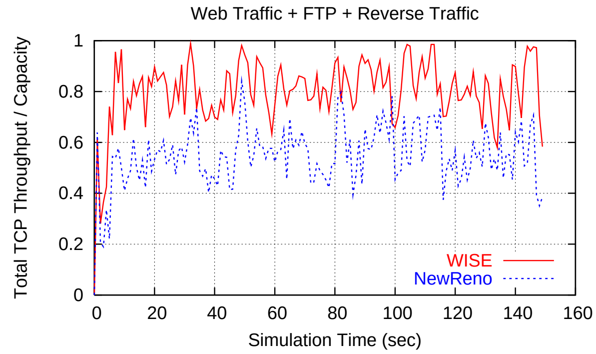

Fig. 9. WiSE boosts performance over wireless links

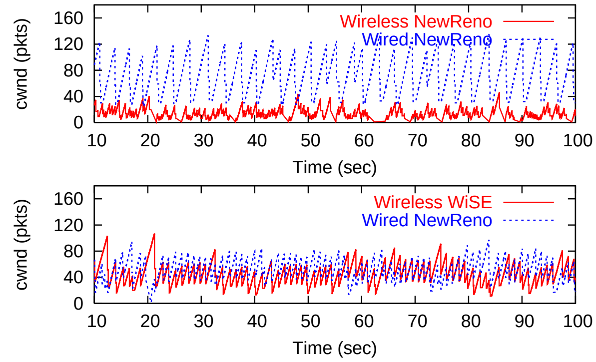

Fig. 10. A wired & wireless flows sharing a congested RED+ECN router; (top) both flows are NewReno; (bottom) the wireless flow is WiSE.

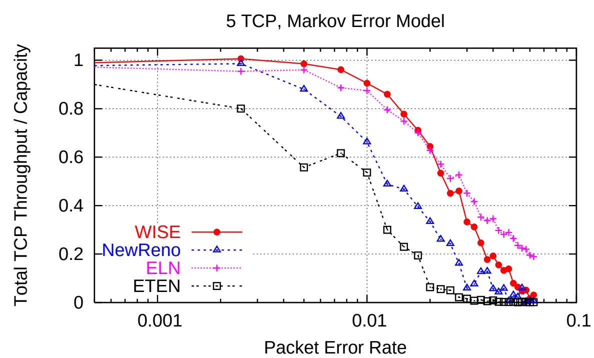

Fig. 11. Comparison of WiSE, NewReno, ELN and ETEN

(D) Comparison of WiSE, NewReno, ELN, and ETEN: Figure 11 shows the average combined throughput of 5 long-lived TCPs traversing a 10 Mb/s wireless link as a function of the packet error rate. It shows that WiSE achieves a performance gain comparable to ELN and better than ETEN, without generating extra messages or requiring dedicated header space. ETEN [18] sends out-of-band ICMP messages to the sender to report the packet corruption rate at the wireless link. ELN [2] assumes route symmetry and uses a dedicated field in the TCP header to report the corruption of a certain packet.

Multiple Wireless Links:
So far we have seen that WiSE can improve the performance of TCP sources over a single wireless link. In this section, we study how WiSE performs when there are multiple wireless links, which is used to represent a small all-wireless network. A network configuration with six nodes is constructed as shown in Figure 12. All links are wireless, with capacity of 10Mbps, and have a latency of 10ms. We use the Markov error

model throughout this section. There are three long-lived TCP flow groups (5 flows in each) in this network. Group $grp1$ sends from node $n1$ to node $n6$, group $grp2$ sends from node $n4$ to $n2$, and group $grp3$ sends from node $n3$ to node $n6$.

Note that the links between $n1$-$n2$ and $n4$-$n5$ are congested. Figures 13 and 14 show the performance with and without WiSE. It shows that WiSE results in a substantial improvement in link utilization.

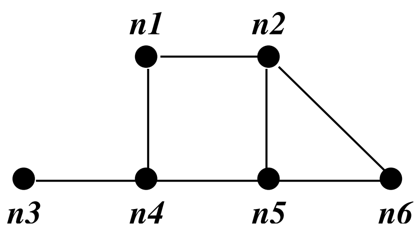

Fig. 12. A small all-wireless network

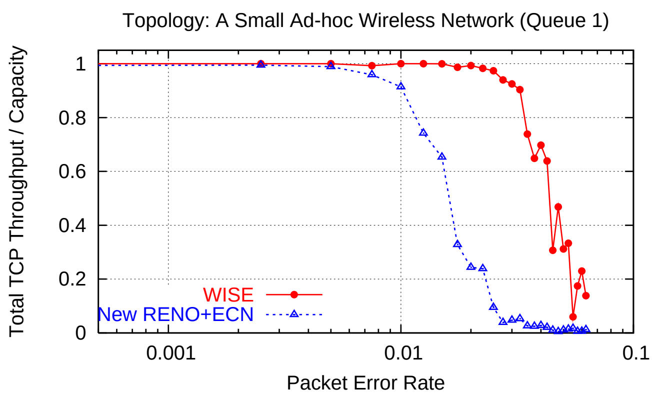

Fig. 13. Aggregate TCP throughput in a small all-wireless network: Link $n1$-$n2$.

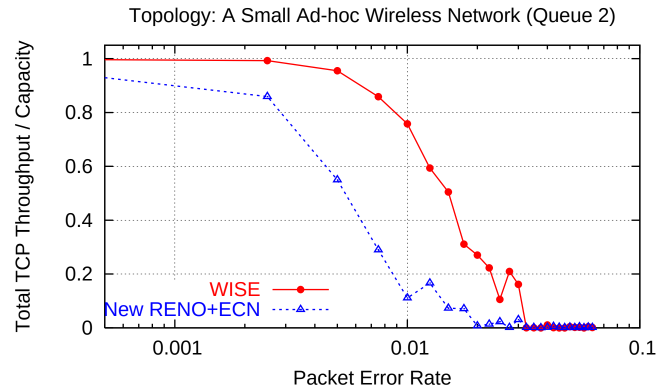

Fig. 14. Aggregate TCP throughput in a small all-wireless network: Link $n4$-$n5$.

VII. Some Characteristics of M-ECN and WiSE

A. Security & Interaction with Nonce

To prevent a receiver from ignoring ECN congestion marks, the sender sets a one bit random nonce in each packet. The receiver returns the nonce sum to the sender in the Acks. But when a router marks a packet it clears the nonce, and the receiver would not know the correct nonce sum [25].

Communication on the M-ECN channel works properly when nonce are used. Further, WiSE can use the nonce to prevent a receiver from cheating by making a congestion drop look as if it were a corruption drop. Since the WiSE router can read the value of nonce before marking a packet, it can help the receiver adjust its nonce sum, and prove to the sender the correctness of a WiSE mark. In particular, the WiSE router maintains a single bit called Δ-nonce in which it stores the cumulative sum of the nonce in all packets that got corrupted or marked during the encoding of a single WiSE codeword. The Δ-nonce is added to the nonce value in the first unmarked packet after the codeword. This causes the receiver's nonce sum to match that of the sender as long as there are no congestion drops/marks. Said differently, the receiver cannot claim that a congestion drop is a corruption drop because, without the help of a WiSE agent, the receiver doesn't know the nonce value of the dropped packet.

B. Robustness to Packet Reordering

Packet reordering does not flip 1's to 0's and thus does not lead to the loss of congestion marks. Reordering is unlikely to create legal M-ECN codewords because codewords are well-structures patterns, which are hard to generate by random reordering. Reordering may destroy an M-ECN codeword causing the loss of the M-ECN signal. For example, in WiSE, reordering could cause X10 $\rightarrow$ X01; thus TCP will mistake the error drop to be a congestion drop, which is what would have happened anyways if we didn't use WiSE. Further, since reordering occurs between outstanding packets, the reordered WiSE marks don't do any harm because TCP reacts to only one congestion signal per congestion window.

C. Partial Deployment

The M-ECN channel is amenable to partial deployment because it is completely transparent to routers and flows which do not participate in the communication. For example, in §VI-C, we have shown that WiSE works well when the path traverses both WiSE and non-WiSE routers. We have also shown that WiSE functions correctly independently of whether the bottleneck is a wired or wireless link. Finally, we have also shown that WiSE interacts properly with non-WiSE TCPs.

VIII. XECN: eXtended ECN

As an alternate application of the M-ECN channel, we develop a scheme called XECN which helps flows with a very large end-to-end pipe to both quickly acquire spare bandwidth and also refine their decrease signal. We believe that the number of gigabit flows over a path will be small and thus, a per-gigabit-flow state might be acceptable. M-ECN is attractive for this application because it doesn't require all routers along the path to understand the new signaling. When legacy routers along the path generate ECN congestion marks/drops, they create an illegal M-ECN codeword. This causes the sender to fall back to traditional TCP and to halve its congestion window. Our aim in this section is only to provide proof of concept for another example of network-to-transport signaling, and therefore we will explain XECN at the algorithmic level without going into implementation details and evaluating the scheme. These issues will be the focus of future work.

A. Overview of XECN

Standardly, packet drops/marks provide a binary indication of congestion and the lack thereof is assumed to imply that a flow can increase its window size by one packet per RTT. Thus, the decrease signal is binary and there is no increase signal in traditional TCP. An example of network-to-transport signaling is therefore the communication of finer information on both the presence and lack of congestion.

Instead of only conveying a binary indication of congestion, XECN translates the level of congestion into a feedback value. While we do not answer the question: "What form of feedback is most appropriate?", for concreteness, assume that the decrease signal corresponds to the fraction by which the router wants the sender to reduce its congestion window whereas the increase signal is given in terms of an absolute number of packets by which a flow can increase its `cwnd_`. In this section, we show how to send this information from the routers to the sender over the M-ECN channel.

The XECN signal will be encoded over a fixed-size frame of packets. Hence, unlike WiSE and akin to traditional block codes for data transmission, XECN codewords are of fixed length. Below we describe the information coding scheme used by the router to convey the fine-tuned congestion feedback to the gigabit flows. We assume the router runs a controller that decides the feedback value. The exact computation of the feedback is beyond the scope of this paper; here we only show how to convey this multi-bit feedback to the sender using the M-ECN channel. The implementation of XECN is a bit more involved than WISE so we will devote a section each for the different aspects of the scheme.

B. Frame syntax

In the description ahead, whenever we refer to an ECN bit having the value *, we actually mean that both the ECT and CE bits in the IP header of that packet are 0. This design choice of using a * instead of an ECN = 1 has the following advantage: it ensures that no ECN congestion marks are misunderstood to be XECN marks. This is a possibility that we cannot rule out if we use 1's instead of *'s.

An XECN frame will be 40 bits long, i.e it will span 40 packets. It will consist of a **Preamble** which is the string "*000*". The Preamble is followed by a lone * at some position in a string of length 35. The position of the * will signal by what fraction (value) the router would like the source to decrease (resp., increase) its window size `cwnd_`.

The choice of 40 as frame size is partly motivated[4] by the following statistical consideration: since we would like to minimize the effect of XECN marks on the normal functioning of ECN, we would like that the percentage of *'s set by XECN be as small as possible. This is because in time of congestion, an AQM router cannot ECN-mark packets marked with a * to signal congestion and has to drop. We can achieve by having a large frame length, but that would increase both control latency and probability of incorrect decoding of the XECN signal. Thus 40 is a compromise value which along with the 3 *'s (two for the preamble and one for the signal) translates to a figure of 7.5% ECN bits being set to *. It has been observed that RED has an overall marking probability of about 1%. This, in conjunction with our design figure, leads to a low probability of 0.075% of a RED ECN mark clashing with a * and thus causing a packet drop. Thus, by using *s instead of 1s to create XECN codewords, we are very conservative and resilient to the loss of a congestion signal. The disadvantage, however, is that sometimes, to convey a congestion signal, the AQM router has to drop a packet with ECN=*, though the router could have marked the packet if the packet were declared ECN-enabled. This happens with low probability because most of the packets are ECN-enabled. Further, dropping a packet (instead of marking it) does not cause that much wastage of resources since we are operating in a high bandwidth environment.

A frame is depicted in Figure 15, with 5 bits of the **PREAMBLE** followed by 35 bits of the signal.

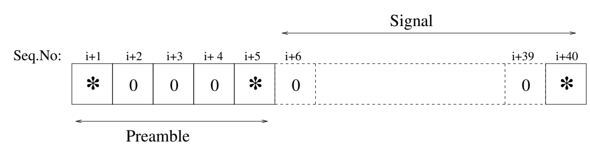

Fig. 15. An XECN frame

We will now explain how to interpret the lone * in the signal portion of the frame. To signal to the source that it should increase/decrease its `cwnd_` we must signal to it the proportion by which to make the change. This is achieved by dividing the signal portion of the frame into two parts about position 23. All bits to the left will signal fractions of decrease on some linear scale from 0.5 (corresponding to a maximum decrease by halving `cwnd_`) to 1 (corresponding to no decrease). All the bits to the right will signal an absolute increase in the window size, again on a linear scale from 0 (no increase) to some predecided maximum increase inc_{max}. Both scales will be represented by a bit present in that position from left to right, with position 23 being called **Neutral** because it corresponds to no change (1 on the decrease scale and 0 on the increase scale coinciding with it). This coding scheme is shown in Figure 16.

[4]We also keep in mind that in the HighSpeed TCP proposal, the Low Window Size is defined to be 38.

C. The encoding scheme

A sender subscribes to the XECN service only after its window size becomes large enough to ensure that it can get a whole XECN codeword across in one RTT. (Subscription is done using the IP router alert option.) Thus, having chosen the frame size to be 40, it means that a sender will wait till it's window size is 40, and then after say 5 RTTs subscribe to XECN by an IP option. At this time, every flow starts to repeatedly send the following default codeword in the ECN bit stream: *000*00000000000000000000000* (i.e., the sender asks for maximum increase). At the XECN router, the XECN bit position `X_Pos` is calculated as a function of the congestion level. The job of the ecnoder (not a typo) is to embed the congestion feedback from the router using X_Pos. It first waits for a valid preamble to pass through the queue. If it observes the sequence *000* in the ECN bits of consecutive packets from a flow, it assumes that a frame has begun. Then the ecnoder probes every subsequent packet: if a packet before the one on which it has to mark the XECN bit has its ECN bit set then the ecnoder ignores its signal for the current frame. Otherwise it sets the packet at X_pos to *. In this case it erases the next * it sees before the preamble of the next frame.

Note that every router along the path computes its feedback and scans the stream of ECN bits to see whether the feedback coming from upstream is larger (i.e., more increase of less decrease). If the feedback coming from upstream is larger then the router overwrites it by moving the * to the position that conveys the feedback at this router. If the feedback coming from upstream is smaller (i.e., more decrease or less increase), the router let it go through with no modification because some upstream router is more congested. The router can scan the stream of ECN bits and update it in the same time because decreasing the feedback means moving the star to a later packet.

The particular encoding strategy can be justified by keeping in mind one of our design objectives: to minimize harm to the normal functioning of the ECN scheme. We achieve this goal by trying to conserve the number of ones. Thus a bit set to * due to an ECN mark remains that way right till the corresponding ACK reaches the sender. The decoder at the source can then spot that there is an extra one in a frame. Since a decision to cut the congestion window by half dominates all other signals from other routers along the path of a flow, an extra one immediately leads XECN to switch to ECN mode for that frame. In the situation where there is no ECN mark in a frame, the harshest decrease signal dominates, hence the one that is marked earliest implies that a one seen ahead in the frame has to be erased in order to conserve the number of ones.

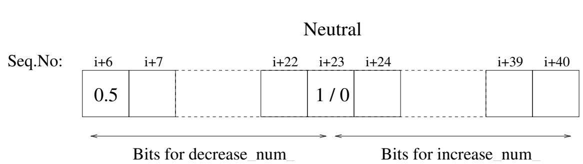

Fig. 16. Encoding the signal in an XECN frame

D. The decoding algorithm

Roughly speaking, the decoder works as follows: it first tries to detect a valid preamble. If successful, it waits for the next * to appear. The decision to act on a decrease signal is taken differently from that of an increase signal. This fundamental difference is because we wish to be conservative in our actions, owing to the fact that in the face of drops corrupting our frames and the possibility that genuine ECN marks may exist in a frame, a rash increase action will harm the system performance more significantly than if the source were to cut its sending rate. In addition, the coding scheme is such that it allows for a decrease signal to be decoded early on to facilitate immediate action.

Thus, if a * appears before position **Neutral** then it is acted upon immediately in the form of a congestion window decrease by the appropriate fraction. This step is correct because a one is either an XECN or an ECN mark. If it is the former, then the source should react to the earliest (largest) decrease signalled. But if the one is actually an ECN mark, then in all probability another one will be seen by the decoder in the same frame. The decoder always assumes that if there are more than 3 ones in a frame then an ECN mark has been placed so that the source will decrease its window size by more so that overall `cwnd_` is halved. In any case TCP doesn't react to more than one drop/mark in a round-trip time (RTT), so no special action is required if the number of extra marks is more than one.

On the other hand, if the first post-preamble * in a frame occurs after position **Neutral**, implying if it were actually an XECN mark that the source should increase its `cwnd_`, then we store this information. We never act upon an increase until we have decoded the preamble of the next frame. This is to verify that it is indeed an XECN increase signal. If another one is encountered before the next preamble, then an ECN mark is implied and we half the congestion window. If the next preamble doesn't arrive at the expected end of frame then the validity of the supposed XECN increase signal cannot be verified, and that mark is ignored.

IX. Conclusion and Future work

This paper describes the use of ECN bit streams for signaling extra information *without* interfering with the basic functionality of ECN. We have developed the M-ECN channel, studied it information theoretically using the **ZwE**-channel, and determined that ECN signaling only uses a portion of the capacity of the M-ECN channel. We then applied the M-ECN channel abstraction to an important problem in wireless networks: signaling transmission loss notification to TCP sources.

Specifically, we developed the WiSE protocol and simulations showed that its performance is very competitive with respect to previous solutions both over a single wireless channel and over multiple wireless links. Notably, the throughput it delivered was higher and flatter over a larger range of packet error rates. WiSE introduced a simple encoding (decoding) routine at the router (respectively, the user). Unlike previous solutions, WiSE doesn't need extra space in IP headers and doesn't generate extra traffic.

We also proposed and gave an outline of an application of M-ECN to congestion control for large flows, namely XECN. Further work is required to compute a feedback value that will lead to an efficient allocation of resources and stable dynamics for the network. We also observed that different applications led to very different coding mechanisms for signaling over the M-ECN channel

Future work consists of exploring the use of the M-ECN channel to signal other information, e.g. other path characteristics such as the typical drop rate or queuing delay which could be used by streaming media applications. An interesting issue is to consider the question of a user who wants two different types of information signaled to it by the network using the same channel (i.e. over ECN bit streams). While the encoding/decoding scheme we have advanced in WiSE is adequate for coping with channel errors and is simple to implement, it is worth considering the design of such schemes more systematically. This may lead to higher signaling rates and even simpler implementations.

References

[1] A. Bakre and B. R. Badrinath. I-TCP: Indirect TCP for mobile hosts. In *Proc. 15th International Conference on Distributed Computing Systems (ICDCS)*, May 1995.

[2] H. Balakrishnan and R. Katz. Explicit loss notification and wireless web performance. In *nms.lcs.mit.edu/papers/globecom98/*, Nov. 1998.

[3] H. Balakrishnan, S. Seshan, and R. Katz. Improving reliable transport and handoff performance in cellular wireless networks. In *ACM Wireless Networks*, Dec. 1995.

[4] D. Barman and I. Matta. Effectiveness of loss labeling in improving TCP performance in wired/wireless networks. Technical report, Boston University, 2002.

[5] S. M. Bellovin. Security problems in the TCP/IP protocol suite. In *Computer Communications Review 2:19, pp. 32-48*, Apr. 1989.

[6] S. Biaz and N. H. Vaidya. Distinguishing congestion losses from wireless transmission losses: A negative result. *Seventh International Conference on Computer Communications and Networks (IC3N)*, 1998.

[7] T. M. Cover and J. A. Thomas. Elements of information theory. In *Wiley Series in Telecommunications*, 1991.

[8] D. C. Cox. Wireless personal communications: what is it? In *IEEE Personal Communications*, volume 2 Issue:2, pp 20-35, Apr. 1995.

[9] S. Dawkins, C. E. Williams, and A. E. Yegin. Problem statement for Triggers for Transport(TRIGTRAN). In *http://www.ietf.org/internet-drafts/draft-dawkins-trigtran-probstmt-00.txt*, Oct. 2002.

[10] D. Duchamp and N. F. Reynolds. Measured performance of a wireless lan. *Proceedings of the 17th Conference on Local Computer Networks, pages 494-499*, Sept. 1992.

[11] D. A. Eckhardt and P. Steenkiste. Measurement and analysis of the error characteristics of an in-building wireless network. *Proceedings of SIGCOMM '96, Stanford, CA*, Aug. 1996.

[12] D. A. Eckhardt and P. Steenkiste. Improving wireless LAN performance via adaptive local error control. *Proceedings of IEEE ICNP '98*, 1998.

[13] Alternate proposals for the ECN field in IP, or for other ECN-like semantics. http://www.icir.org/floyd/ecn.html.

[14] S. W. Golomb. The limiting behavior of the z-channel. In *IEEE Transactions on Information Theory*, volume IT-26 No. 3, May 1980.

[15] D. Katz. IP Router Alert Option. *RFC 2113*, Feb. 1997.

[16] R. H. Katz. Adaptation and mobility in wireless information systems. In *IEEE Personal Communications*, volume 1 Issue:1, pp 6-17, Apr. 1994.

[17] S. Kent and R. Atkinson. Security architecture for the internet protocol. *RFC 2401*, Nov. 1998.

[18] R. Krishnan, M. Allman, C. Partridge, and J. P. G. Sterbenz. Explicit transport error notification for error-prone wireless and satellite networks. In *BBN Technical Report No. 8333*, Mar. 2002.

[19] R. Mahajan, S. M. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. In *Computer Communications Review 32:3, pp. 62-73*, July 2002.

[20] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang. TCP Westwood: Bandwidth estimation for enhanced transport over wireless links. In *Mobile Computing and Networking*, pages 287–297, 2001.

[21] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. Tcp selective acknowledgement options. In *RFC 2018*, Apr. 1996.

[22] G. T. Nguyen, B. Noble, R. H. Katz, and M. Satyanarayanan. A trace-based approach for modeling wireless channel behavior. In *Proc. of the 1996 Winter Simulation Conference*, Dec. 1996.

[23] The network simulator ns-2. http://www.isi.edu/nsnam/ns.

[24] P. Oprisan and B. Bose. ARQ in Optical Networks. In *Proc. Pacific Rim International Symposium on Dependable Computing*, Dec. 2001.

[25] K. K. Ramakrishnan, S. Floyd, and D. Black. The addition of explicit congestion notification (ECN) to IP. *RFC 3168*, Sept. 2001.

[26] T. S. Rappaport. Wireless communications: Principles and practice, second edition, prentice hall. 2002.

[27] S. Savage, D. Wetherall, A. Karlin, and T. Aderson. Practical network support for IP traceback. In *Proc. of ACM SIGCOMM*, 2000.

[28] I. Stoica, S. Shenker, and H. Zhang. Core-stateless fair queuing: A scalable architecture to approximate fair bandwidth allocations in high speed networks. In *Proc. of ACM SIGCOMM*, Aug. 1998.

X. APPENDIX

The entropy of a discrete random variable X with a probability mass function described by the vector $(p_1, ..., p_n)$ is given by:

$$H(X) = H(p_1, ..., p_n) = -\sum_{i=1}^{n} p_i \log_2 p_i.$$

A discrete channel takes inputs from a discrete alphabet X and maps them to outputs from a discrete alphabet Y through a channel probability matrix $P(y|x)$ for all $x \in X$ and $y \in Y$. A discrete channel is called memoryless if given the current input, the current output is independent of all past inputs and outputs. The capacity of a discrete memoryless channel is defined to be:

$$C = \max_{0 \leq q \leq 1} I(X : Y),$$

where $I(X;Y)$ is the mutual information between the input X and the output Y, and the maximum is over all possible probability distributions on the input. Since the **ZwE**-channel has a binary input $\{0, 1\}$ and output alphabet $\{0, 1, e\}$, the optimization is over $q = P(X = 1) = 1 - P(X = 0)$. As mentioned before, $I(X;Y) = H(Y) - H(Y|X)$, where $H(Y)$ is the entropy of the output and $H(Y|X)$ is the conditional entropy of the output given the input. For the **ZwE**-channel, we calculate

$$H(Y) = H\left(\epsilon, q(1-\epsilon) + (1-q)p, (1-q)(1-p-\epsilon)\right)$$

and

$$H(Y|X) = qH\left(\epsilon, 1-\epsilon\right) + (1-q)H\left(\epsilon, p, 1-p-\epsilon\right)$$

which gives us $I(X;Y)$ as a function of p and ϵ. We obtain the optimizing value q^* by differentiating $I(X;Y)$ and equating to zero. Thus we get

$$\frac{\bar{q}}{\bar{\epsilon} - \bar{q}(\bar{\epsilon} - p)} = \left(\frac{p^p}{\bar{\epsilon}^{\bar{\epsilon}}}\right)^{\frac{1}{\bar{\epsilon}-p}}$$

where $\bar{q} = 1 - q$ and $\bar{\epsilon} = 1 - \epsilon$. On simplification, this gives us

$$q^* = \frac{(1-\epsilon)^{\frac{1-\epsilon}{1-p-\epsilon}} - p^{\frac{1-\epsilon}{1-p-\epsilon}}}{(1-\epsilon)^{\frac{1-\epsilon}{1-p-\epsilon}} - p^{\frac{1-\epsilon}{1-p-\epsilon}} + (1-\epsilon)p^{\frac{p}{1-p-\epsilon}}}.$$

Thus for the **ZwE**-channel, the capacity as a function of p and ϵ is:

$$C(p, \epsilon) = I(X : Y)|_{q=q^*}.$$

Now, the mutual information is a concave function, and monotonically increasing for $q \in [0, q^*]$, so if we set a constraint on the density of ones, say $q \leq q'$ for some $q' \leq q^*$ then the capacity is $I(X;Y)|_{q=q'}$. This is the quantity plotted in Figure 4.