

Network Coding Research Group (NCWCRG)
Internet-Draft
Intended status: Informational
Expires: January 16, 2014

C. Adjih
Inria
SY. Cho
Samsung Electronics Co.,LTD.(*)
E. Baccelli
Inria
July 15, 2013

Broadcast With Network Coding: DRAGONCAST
draft-adjih-dragoncast-00

Abstract

This document describes a Network Coding (NC) based broadcast protocol suitable mainly for wireless networks, including mobile ad hoc networks (MANET). It provides data dissemination from a single source, based on intra-flow network coding and for Internet Protocol (IP). It is designed to minimize the assumptions over the working environment and thus may operate over dynamically evolving environments.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 16, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology	4
3. Protocol Overview	5
3.1. General Functioning	6
3.2. Protocol Principles and Definitions	7
4. DRAS: DRAGONCAST Signaling	8
5. DRALIB: DRAGONCAST Local Information Base	11
6. DRAGONCAST Protocol Functioning	13
6.1. Source Packet Generation	14
6.2. Packet Processing	14
6.3. Packet Generation	14
7. DRAGON: Packet Rate Selection	14
7.1. DRAGON Rationale	14
7.2. DRAGON (Dynamic Rate Adaptation from Gap with Other Nodes)	15
8. Real-time Decoding: Sliding Encoding Window (SEW)	16
9. Security Considerations	18
10. IANA Considerations	18
11. Informative References	18
Appendix A. Network Coding Fundamentals	19
A.1. Linear Coding	19
A.2. Random Linear Coding	20
A.3. Decoding, Vector Space, and Rank	20
Appendix B. Acknowledgements	21

1. Introduction

The goal of this document is to describe a protocol for broadcast in wireless networks with network coding.

It is best suited to multi-hop wireless networks. Compared to wired networks, they have specific properties, see for instance [I-D.baccellli-multihop], including:

- Wireless 'neighborcast': one wireless transmission by a node may reach several receivers. This property may be used to optimize broadcast.
- Time-variation: the visibility between two nodes may evolves with time, due to node mobility, physical changes in the propagation environment or other reasons.
- Unreliability of wireless communications: due to wireless channel conditions or properties, transmissions losses (packet erasures) potentially occur.

In wireless networks, applications sometimes require the communication primitive of broadcasting information to the entire network. As an example, in the context of MANETs, the Simplified Multicast Forwarding (SMF) [RFC6621] has been proposed for this purpose.

For applications where a single source sends a large volume of information to the entire network, it will be split in several packets, which will then be broadcast to the entire network. In this case, network coding can provide two features: natural error correction, and efficiency (in terms of the number of transmissions). A large literature on this topic has evidenced the benefits of network coding and explored design and implementation aspects.

In this document, we present DRAGONCAST (D.R.A.G.O.N., "Dynamic Rate Adaptation from Gap with Other Nodes"), one such protocol for broadcasting with network coding.

It is based on intra-flow coding. It attempts to maximize simplicity and universality: it does not use explicitly or implicit knowledge relative to the topology (such as the direction or distance to the source, the loss rate of the links, ...), hence is perfectly suited to the most dynamic wireless networks.

2. Terminology

Abbreviation	Definition
MANET	Mobile Ad Hoc Network
NHDP	Neighborhood Discovery Protocol
OLSR	Optimized Link State Routing
SMF	Simplified Multicast Forwarding
TLV	Type-Length-Value encoding
DRAGON	Dynamic Rate Adaptation from Gap with Other Nodes
SEW	Sliding Encoding Window
PME	Protocol Message Element
F-PME	Flow Protocol Message Element
S-PME	State Protocol Message Element
ED-PME	Encoded Data Protocol Message Element

Table 1

The following table summarizes the definitions and notations related to network coding in general and to DRAGONCAST. More details on network coding can be found in Appendix A.

Name	Definition
Coded Packet	Linear combination of source packets
Encoding	Coefficients in the linear combination of source packets
Vector	abbrev. for "Information Vector", coded packet
Innovative	A coded packet that brings new information

Table 2

Name	Notation
Source packets	$P(1), \dots, P(k)$
Linear combination of packets	$a(1) P(1) + \dots + a(k) P(k)$
Set of coded packets at a node v	$q(v,i) = a(v,i,1) P(1) + \dots + a(v,i,M) P(M)$

Table 3

3. Protocol Overview

DRAGONCAST is a protocol for broadcasting a set of packets from one source to a entire network with network coding (various parts are described in [CA08a], [CA08b], [C08]). The protocol is distributed and requires minimal coordination.

The base functioning is simple: the broadcast is initiated by transmissions for the source. Every node in the network retransmits coded packets with a changing interval between transmissions. At the same time, every node collects received coded packets, and performs decoding as they are received. Finally, termination is automatically detected when all the nodes have successfully received all data.

DRAGONCAST is based on several building blocks that are in two categories. The first category concerns the protocol aspects themselves:

- o DRAS (DRAGoncast Signaling): the signaling for the control plane for DRAGONCAST. The signaling is mostly present on a header to each coded packet (e.g piggybacking). It includes with information relative to the state of the node, in addition to the packet encoding information. This allows each node to maintains information about the state of its neighbors.
- o DRALIB (DRAGoncast Local Information Base): this information base maintains information about the flows, the decoding process, and the state of the neighbors
- o DRAGONCAST: the protocol itself with message generation and message processing

The second category is relative to policy, that is building blocks that define high level protocol behavior:

- o DRAGON: a dynamic packet rate adjustment policy. Every node is retransmitting coded packets with a certain packet rate; this rate is adjusted dynamically. Essentially, the rate of the node increases if it detects some nodes in the current neighborhood are "falling behind" in the decoding process. This is called a "dimension gap", and the proposed adaptation algorithm is a Dynamic Rate Adjustment from Gap with Other Nodes (DRAGON), a heuristic.
- o SEW: a real-time decoding method denoted SEW (Sliding Encoding Window). It does not requires the concept of generation; instead, the knowledge of the state of neighbors is used to constrain the content of generated coded packets and allow real-time decoding.

SEW maintains a buffer of the currently undecoded packets.

The Figure 1 represents the organization of the different building blocks.

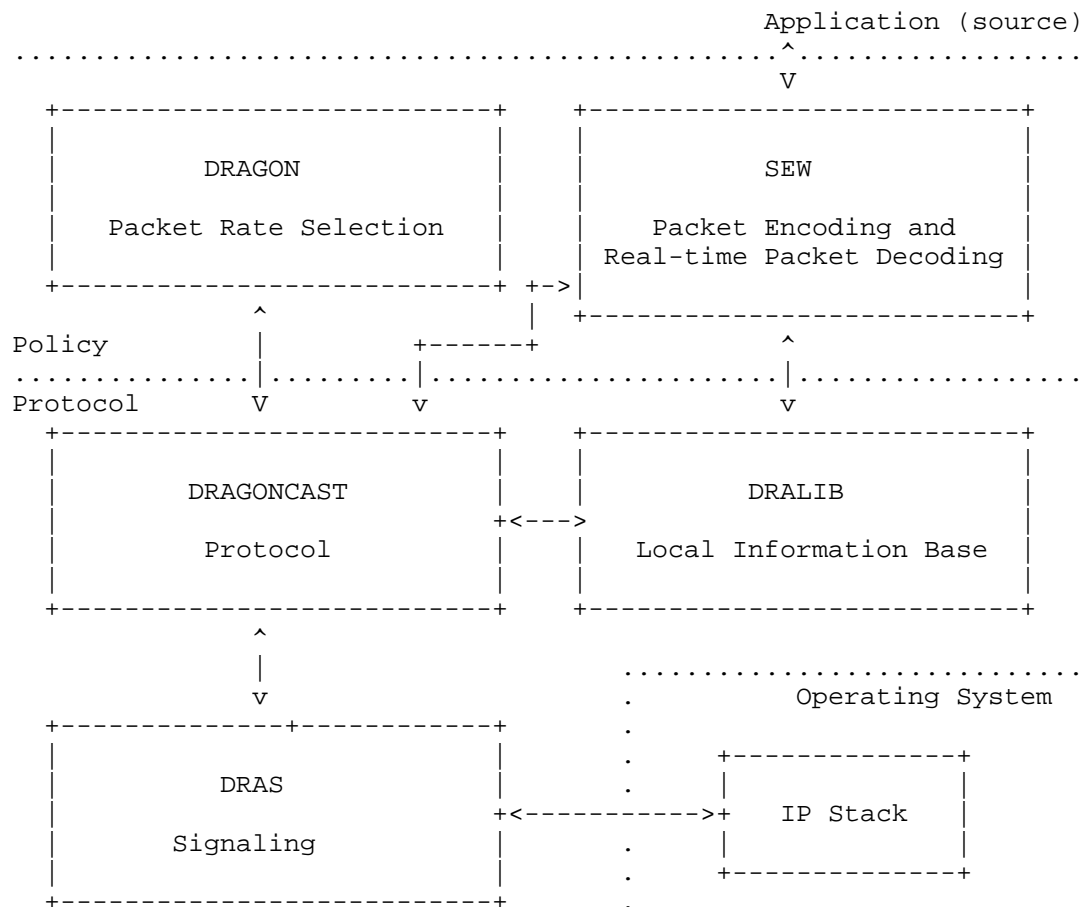


Figure 1: Organization of the DRAGONCAST Building Blocks

3.1. General Functioning

The source initiates broadcasting by sending its original data packets with a format specified in Section 4.

When the source sends data packets, it produces packets of a predefined, constant size, using padding if necessary. Other nodes initiate transmission of encoded data upon receiving the first coded packet, and stay in a transmission state where they will retransmit

coded packets with an interval decided by packet rate selection algorithms. Precisely, when intermediate nodes receive a data packet that is a source packet or a coded packet, they start scheduling encoded data transmission.

The scheduling interval is decided by the policy DRAGON from Section 7. Transmitted packets by intermediate nodes are coded packets generated using random linear coding with a header specified in Section 4. Data transmission continues until nodes detect the termination condition, i.e. that themselves and all their neighbors have successfully decoded the data stream.

General operations of the protocol are described in the following algorithm:

1. Source data transmission scheduling: the source transmits sequentially D vectors (packets) of a generation with rate C
2. Nodes' data transmission start condition: the nodes start transmitting a vector when they receive the first vector.
3. Nodes' data storing condition: the nodes store a received vector in their local buffer only if the received vector has new and different information from the vectors that the nodes already have.
4. Nodes' termination conditions: the nodes continue transmitting until themselves and their current known neighbors in their local information base have enough vectors to recover the D source packets.
5. Nodes' data transmission scheduling: every node retransmits linear combinations of the vectors in its local buffer after waiting for a delay computed from the rate selection.
6. Nodes' data transmission restart condition: When one node receives a notification indicating that one neighboring node requires more vectors to recover the D source packets and it has already stopped data transmission, the node re-enters in a transmission state.

3.2. Protocol Principles and Definitions

DRAGONCAST uses the following concepts and definitions:

- o Source: a source is a node that will broadcast information to the network. This information is called a "flow". A source may have an arbitrary number of flows, however each flow will be coded

independently (intra-flow coding).

- o Flow: a flow is the unit of transmission of the protocol DRAGONCAST. The flow represents a sequence of bytes at the source that need to be broadcast. The source divides the flow in a sequence of packets of equal size (padding may be used). The packets are numbered, and can be identified by their packet index. The packets of one flow may optionally divided in several generations (one per default).
- o Generation: a generation is a subset of a consecutive packets of one flow.

4. DRAS: DRAGONCAST Signaling

DRAGONCAST uses one single packet format based on a sequence of Type-Value including several protocol elements. The general packet format is represented in Figure 2.

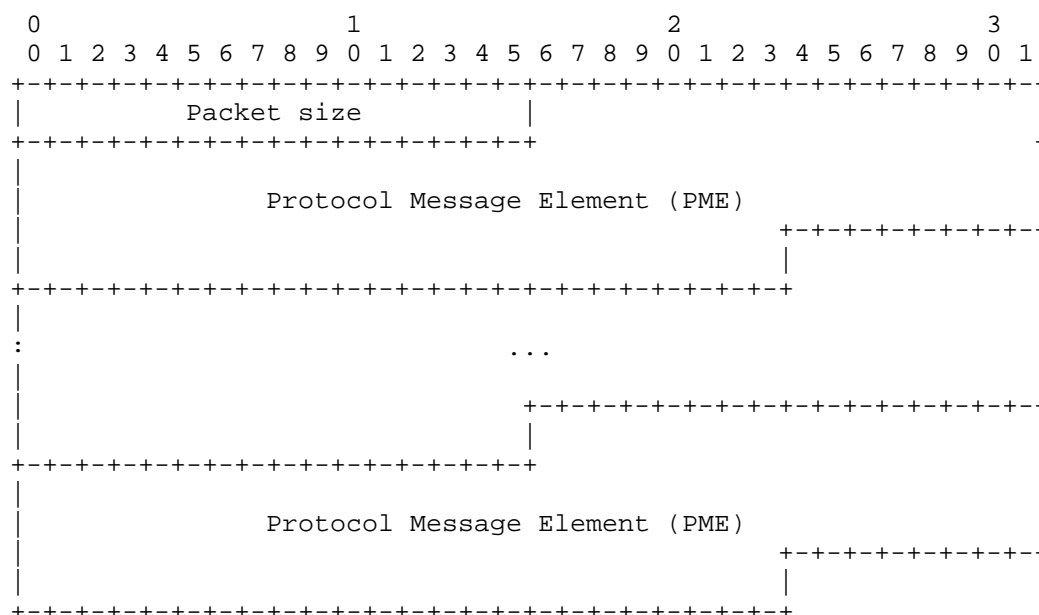


Figure 2

The following protocol elements are defined:

- o The Flow Protocol Message Element (F-PME), Figure 3: it specifies information identifying the flow and associated (constant) parameters.

- o Source Packet Rate: expressed as the average inter-departure of the coded packets in milliseconds (e.g. "10 packets per second" yields the value 100).
- o Generation Number of Packets: total number of packets in the Generation with the given Generation ID.
- o Sliding Encoding Window Size: the size of the encoding window, when generating coded packets.

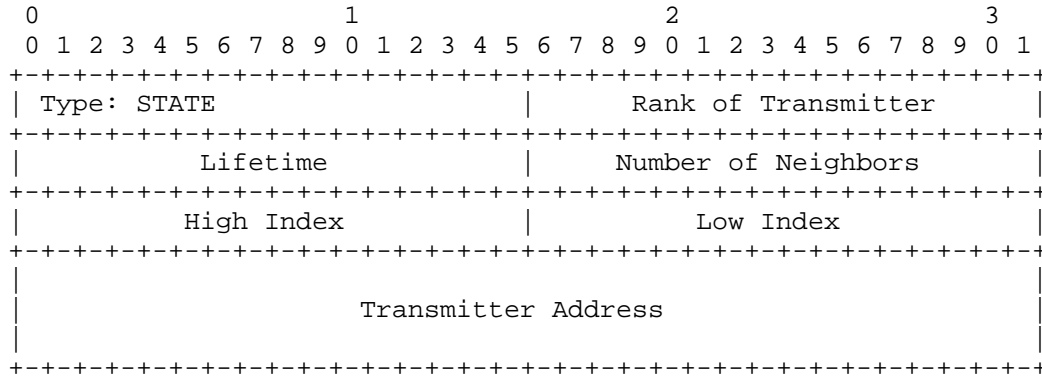


Figure 4: Format of the State PME

The State PME specifies state information of transmitter with respect to the generation identified by a preceding F-PME and includes:

- o Rank of Transmitter: denotes the current amount of innovative data of the transmitter for the generation.
- o Lifetime: denotes duration during which the information in this packet (that is, the rank and the fact that transmitter is a neighbor of a node receiving this packet) is considered valid (after this it will expire).
- o Number of neighbors: denotes the number of neighbors heard, that are not yet expired.
- o High Index: specifies the highest index present in a undecoded linear combination in the decoding table.
- o Low Index: specifies the lowest index present in a undecoded linear combination in the decoding table. Hence all source packets with lower indices have been decoded.

- o Transmitter Address: the IP address of the transmitter of the message.

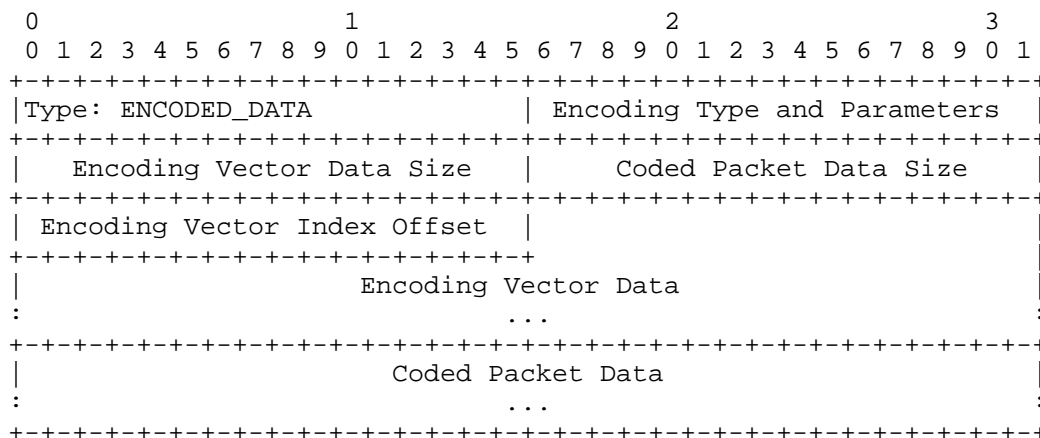


Figure 5: Format of the Encoded Data PME

The Encoded Data PME holds actual coded packet content:

- o Encoding Type and Parameters: in this version of the specification, the fields contains one of the constants ENCODING_GF_2, ENCODING_GF_4, ENCODING_GF_16 ENCODING_GF_256. This represents the fact that the fields GF(2), GF(4), GF(16), or GF(256) respectively are used as basis for the linear combination. As a result, the coefficients are respectively coded on 1, 2, 4 or 8 bits.
- o Encoding Vector Data Size: the size of the information representing the encoding vector, in bytes. As indicated above, one byte will hold an integral number of vector coefficients.
- o Coded Packet Data Size: the size of data packets.

Constants: ENCODING_GF_2 = 0 ENCODING_GF_4 = 1 ENCODING_GF_16 = 2
ENCODING_GF_256 = 3

5. DRALIB: DRAGONCAST Local Information Base

A node maintains a Local Information Base that records information about its decoding process, and the state of the neighbors.

The protocol state is maintained per flow: a flow is uniquely identified by a flow identifier. In addition, DRAGONCAST supports

the concept of partitioning a flow into generations. In this version of the specification, each generation is considered as an independent "flow".

The different information bases of DRALIB are structured hierarchically as follows:

- o Flow Information Set. Each flow is independently associated a Flow Information Tuple, which contains one or several generations. Their state is maintained in a:
 - * Generation Information Set. Each generation contains the state of the neighbors with respect to the propagation and the decoding of the generation, stored in a:
 - + Neighbor Information Set, describing the neighbors. Such information may also be provided by another protocol, such as OLSR [RFC3626] or NHDP [RFC6130]

In addition, for decoding purposes, it includes the:

- + Decoding Information Base

Each node maintains a Flow Information Set, which contains collected information about current flows. Specifically, the Flow Information Set consists of Flow Information Tuples each which records:

- o Flow Information Tuple
- o F_flow_identifier: the identifier of the flow
- o F_source_rate: the packet rate of the source
- o F_generation_set: the Generation Information Set associated to the flow

Each node maintains, for each of its Flow Information Tuple, a Generation Information Set, which contains information specific to a generation:

- o Generation Information Tuple
- o G_generation_identifier: an integer (generations are numbered from 0)
- o G_generation_size: number of coded packets in the generation

- o `G_encoding_window_size`: the size of the sliding encoding window (for SEW)
- o `G_decoding`: the Decoding Information Base associated to this generation
- o `G_neighbor_set`: the Neighbor Information Set associated to this generation

For each generation, a node maintains a Neighbor Information Set, which contains its known neighbors (with an expiration time), and information related to their state.

Specifically, the Neighbor Information Set consists of Neighbor Tuples, each of which contain information about a single neighbor, for a given flow and for a given generation, as follows:

- o Neighbor Tuple
- o `N_neighbor_address`: address of the neighbor (heard) node
- o `N_neighbor_rank`: the rank of the neighbor
- o `N_high_index`: high index of the neighbor
- o `N_low_index`: low index of the neighbor
- o `N_validity_time`: the validity time of the tuple (after which it expires)

For each generation, a node maintains a Decoding Information Base with the following content:

- o `D_coded_packet_set`: a set of coded packets. For each, the node maintains:
 - * Encoding vector
 - * Coded packet content

During the decoding process, the decoding module (SEW) performs real-time decoding by performing Gaussian elimination on the list of coded packets.

6. DRAGONCAST Protocol Functioning

6.1. Source Packet Generation

A node that acts as a data source for a flow, also runs a instance of the DRAGONCAST protocol for that flow (e.g. has a Flow Tuple with all associated information).

In addition, it adds periodically source packets in the associated Flow Information Base respecting the source rate.

6.2. Packet Processing

Whenever a node receives an encoded packet:

- o It updates its Flow Information Base related to the associated flow. This includes rank and expiration time.
- o It notifies SEW for real-time decoding. In turn, SEW will forward any decoded packets to the application.
- o It notifies DRAGON which may update the transmission packet rate of the flow.

6.3. Packet Generation

For every "active" flow in its Flow Information Base, a node will generate coded packets, with an interval between packets defined by DRAGON (from the computed packet rate).

If for a given flow and generation, in the associated neighbor set, no neighbor is known to require coded packets, the packet is generated without encoded packet (without Encoded Data PME).

From information in its Local Information Base, every node is able to determines if it needs to sends packets, as described in [C08].

7. DRAGON: Packet Rate Selection

In this section, we describe one packet rate selection algorithms, proposed for DRAGONCAST.

7.1. DRAGON Rationale

The heuristic DRAGON has been proposed and analyzed in [CA08a], and is inspired by Fragouli et al. [FWB06]. We briefly summarize it in this section for completeness. The starting point of our heuristic DRAGON, is that the observation that, for real-time decoding, the rank of nodes inside the network should be close to the index of the last source packet, and that in any case, they should at least evolve

in parallel.

Thus, one would expect the rank of a node to grow at the same pace as the source transmission, as in the example of optimal rate selections for static networks. Decreasing the rates of intermediate nodes by a too large factor, would not permit the proper propagation of source packets in real time. On the contrary, increasing excessively their rates, would not increase the rate of the decoded packets (naturally bounded by the source rate) while it would decrease energy-efficiency (by increasing the amount of redundant transmissions).

The idea of the proposed rate selection is to find a balance between these two inefficient states. As we have seen, ideally the rank of a node would be comparable to the lastly sent source packet. Since we wish to have a simple decentralized algorithm, instead of comparing with the source, we compare indirectly the rank of a node with the rank of all its perceived neighbors.

The key idea is to perform a control so that the rank of neighbor nodes would tend to be equalized: if a node detects that one neighbor had a rank which is too low in comparison with its own, it would tend to increase its rate. Conversely, if all its neighbors have greater ranks than itself, the node need not to send packets in fact.

7.2. DRAGON (Dynamic Rate Adaptation from Gap with Other Nodes)

In this section, we describe the proposed heuristic, DRAGON, as a packet rate selection. It is based on the following definitions. Precisely, let:

- o $D(v,t)$ denote the rank of a node v at time t
- o $N(v,t)$ denote the number of neighbors of one node
- o $g(v,t)$ denote the maximum gap of rank with its neighbors, normalized by the number of neighbors

Then $g(v,t)$ is evaluated as:

- o $g(v,t) = \text{maximum, for all neighbors } u, \text{ of } \{ (D(v,t) - D(u,t)) / N(u,t) \}$

We propose the following rate selection, DRAGON (Dynamic Rate Adaptation from Gap with Other Nodes), which adjusts the rates dynamically, based on that gap of rank between one node and its neighbors as follows: The packet rate of node v is set to $C(v,t)$ at time t as:

- o if $g(v,t) > 0$ then: $C(v,t) = A g(v,t)$ where A is some constant.
- o Otherwise, the node stops sending encoded packets until $g(v,t)$ becomes larger than 0.

When computing the packet rate selection, DRAGONCAST Local Information Base supplies the information necessary: for each neighbor, the rank and the number of neighbors of the last packet received, are maintained in the Neighbor Set. Although they might not necessarily reflect the exact values at the computation time, they are provide an estimate.

8. Real-time Decoding: Sliding Encoding Window (SEW)

In this section, we describe the method of DRAGONCAST for real-time decoding, which allows recovery of some source packets without requiring to decode all source packets at once. It is related to the method from Sundararajan et al. [SSMMB09] described for TCP.

The real-time decoding method, Sliding Encoding Window (SEW) relies on implicit cooperation between neighbor nodes, in order to ensure the possibility of decoding. (Technically, as described in [CA08a], it ensures the existence of a low triangle in global encoding vectors saved in a node during the online Gauss elimination process).

The key of SEW, is to ensure the existence of an early decoding part; to do so, every node in DRAGONCAST keeps track of the state of its neighbors, in order to only send packets, The method SEW relies on two principles:

- o SEW coding rule: generates only coded packets that are linear combinations of the first L source packets, where L is a quantity that increases with time.
- o SEW decoding rule: when decoding, performs a Gaussian elimination, in such a way that one coded packet is only used to eliminate the source packet with the highest possible index (i.e. the latest source packet).

Before detailing the insights behind these rules, we first give definition: the high index of a node, and the low index of a node. As explained in Appendix A, a coded packet is a linear combination of source packets. We define the highest and lowest index of a linear combination, as the minimum and maximum of the coded packets respectively. For instance, a packet $Q = P(3) + P(5) + P(7) + P(8)$, the highest index is 8 and the lowest index is 3.

Because all encoded packets have their own highest index and lowest

index, we can also compute the maximum of the highest index of all not-yet decoded packets in a node, as well as the minimum of the lowest index. We refer to the maximum and the minimum as high index of a node and low index of a node. Notice that a node will generally have decoded the source packets from 1 up to its low index.

The intent of the SEW coding rule is to use knowledge about the state of neighbors of one node, namely their high and low index. A node restricts the generated packets to a subset of the packets of the source, until it is confirmed that perceived neighbors of the node are able to decode nearly all of them, up to a margin K . Notice that once all its neighbors may decode up to the first $L-K$ packets, it is unnecessary for the node to include packets $P(1)$, ... $P(L)$ in its generated combinations.

Hence, the general idea of SEW is that it restricts the mixed original packets within an encoded packet from a window of a fixed size K . In other words, a node encodes only source packets inside a fixed Encoding Window as:

$$\text{i-th generated packet } q(v,i) = a(v,i,k) P(k) + \dots + a(v,i,k_K) P(k+K)$$

where the $\{ P(j) \}$ is the set of packets generated by the source, The sequence of coefficients for $q(v,i)$ is the following global encoding vector: $[0, 0, \dots, a(v,i,k), a(v,i,k+1), \dots, a(v,i,k+K), \dots, 0, 0]$. A node will repeat transmissions of new random combinations within the same window, until its neighbors have progressed in the decoding process.

The intent of the SEW decoding rule, is to guarantee proper functioning of the Gaussian elimination. An example of SEW decoding rule is the following: assume that node v has received packets $q(1)$ and $q(2)$, for instance $q(1) = P(1) + P(9)$ and $q(2) = P(1) + P(2) + P(3)$. Then $q(1)$ would be used to eliminate $P(9)$ for newly incoming packets (the highest possible index is 9), and $q(2)$ would be used to eliminate $P(3)$ from further incoming packets. On the contrary, if the SEW decoding rule was not applied and if $q(1)$ were used to eliminate $P(1)$, then it would be used to eliminate it in $q(2)$, and would result into the computation of $q(2) - q(1) = P(2) + P(3) - P(9)$; this quantity now requires elimination of $P(9)$, an higher index than the initial one in $q(2)$. In contrast the SEW decoding rule guarantees the following invariant: during the Gaussian elimination process, the highest index of every currently non-decoded packet will always stay identical or decrease.

Provided that neighbor state is properly exchanged and known, as a result, the combination of the SEW coding rule and the SEW decoding

rule, guarantee that ultimately every node will be able to decode the packets in the window starting from its lowest index; that is, they guarantee early decoding.

Notice that improper knowledge of neighbor state might impact the performance of the method but not its correctness: if a previously unknown neighbor is detected (for instance due to mobility), the receiving node will properly adjust its sending window. Conversely, in DRAGONCAST, obsolete neighbor information, for instance about disappeared neighbors, will ultimately expire.

9. Security Considerations

This document does not have any security considerations.

10. IANA Considerations

This document does not have any IANA actions.

11. Informative References

- [RFC3626] Clausen, T. and P. Jacquet, "Optimized Link State Routing Protocol (OLSR)", RFC 3626, October 2003.
- [RFC6130] Clausen, T., Dearlove, C., and J. Dean, "Mobile Ad Hoc Network (MANET) Neighborhood Discovery Protocol (NHDP)", RFC 6130, April 2011.
- [RFC6621] Macker, J., "Simplified Multicast Forwarding", RFC 6621, May 2012.
- [I-D.baccelli-multihop] Baccelli, E. and C. Perkins, "Multi-hop Ad Hoc Wireless Communication", draft-baccelli-manet-multihop-communication-02 (work in progress), July 2013.
- [CA08a] Cho, S-Y. and C. Adjih, "Wireless Broadcast with Network Coding: Dynamic Rate Selection", Conference MedHocNet 2008, June 2008.
- [CA08b] Cho, S-Y. and C. Adjih, "Wireless Broadcast with Network Coding: DRAGONCAST", Inria Research Report RR-6569, July 2008.
- [C08] Cho, S-Y., "Efficient Information

- Dissemination in Wireless Multi-Hop Networks", Ecole Polytechnique PhD thesis, Sept 2008.
- [HKMKE03] Ho, T., Koetter, R., Medard, M., Karger, D., and M. Effros, "The Benefits of Coding over Routing in a Randomized Setting", International Symposium on Information Theory 2003, Jun 2003.
- [CWJ03] Chou, P., Wu, Y., and K. Jain, "Practical Network Coding", Forty-third Annual Allerton Conference on Communication, Control, and Computing 2003, Oct 2003.
- [FWB06] Fragouli, C., Widmer, J., and J. Le Boudec, "A Network Coding Approach to Energy Efficient Broadcasting", INFOCOM 2006, Apr 2006.
- [SSMMB09] Sundararajan, J., Shah, D., Medard, M., Mitzenmacher, M., and J. Barros, "Network Coding Meets TCP", INFOCOM 2009, Apr 2009.

Appendix A. Network Coding Fundamentals

This section introduces network coding concepts and terminology.

Network coding differs from classical routing by permitting coding at intermediate nodes. One possible coding algorithm is linear coding that performs only linear transformations through addition and multiplication. Precisely, linear coding assumes identically sized packets and views the packets as vectors on a fixed Galois field. It becomes then possible to perform linear combination of packets.

A.1. Linear Coding

In the case of single source broadcast, all packets initially originate from one source: the source has k packets, which may be seen equivalently as k vectors $P(1)$, ..., $P(k)$ from the vector space over a Galois Field.

With network coding, any coded packet received by node, at any point of time, is a linear combination of some source packets. The i -th received coded packet at a node v is necessarily a linear combination of the source packets $P(1)$, ..., $P(k)$ and can be written as:

$$q(v,i) = a(v,i,1) P(1) + \dots + a(v,i,k) P(k)$$

The sequence of coefficients for a coded packet $q(v,i)$ (denoted "information vector", or simply "vector") is itself a vector $[a(v,i,1), a(v,i,2), \dots, a(v,i,n)]$ (denoted "global encoding vector").

With linear coding, when one node generates a coded packet, it performs a linear combination of the packets that it possesses (the set of received packets $\{ q(v,i) \}$) with some set of coefficients $(g(1), \dots, g(M))$. It then transmits the result:

$$\text{generated_coded_packet} = g(1) q(v,1) + \dots + g(M) q(v,M).$$

Since the vectors $q(v,1), \dots q(v,M)$ are in turn linear combinations of the source packets $P(1), \dots P(k)$, the generated coded packet, can be expressed itself as a linear combination of these, yielding its global encoding vector.

In practice, a special header containing the global coding vector of the transmitted code packet may be added as proposed by Chou et al. [CWJ03].

A.2. Random Linear Coding

As shown above, when a node generates a coded packet with linear coding, an issue is how to select coefficients. Whereas centralized deterministic methods exist, Ho and al. [HKMKE03] presented a novel coding algorithm, which does not require any central coordination. The coding algorithm is random linear coding: when a node transmits a packet, it selects randomly the set of coefficients $\{ g(i) \}$, which will be used to perform a linear combination of the vectors $q(v,M)$ as indicated above.

A.3. Decoding, Vector Space, and Rank

A node v will recover the source packets $P(1), \dots P(k)$ its the received packets $q(v,1), \dots q(v,M)$, considering the matrix of coefficients $a(v,i,j)$ for $i=1, \dots M$ and $j=1, \dots k$ from Appendix A. Decoding amounts to inverting this matrix, for instance with Gaussian elimination.

Thinking in terms of coding vectors, at any point of time, it is possible to associate with one node v , the vector space, $Q(v)$ spawned by the coding vectors, and which is identified with the matrix. The dimension of that vector space, denoted $D(v)$, is also the rank of the matrix. In the rest of this document, by abuse of language, we will call "rank of a node", that rank and dimension.

The rank of a node is a direct metric for the amount of useful received packets, and a received packet is called innovative when it increases the rank of the receiving node. Ultimately a node can decode all source packets when its rank is equal to the the total number of source packets.

Appendix B. Acknowledgements

This document also stems from contributions of and from discussions with : Philippe Jacquet.

Authors' Addresses

Cedric Adjih
Inria

Phone: +33-136-635-215
EMail: Cedric.Adjih@inria.fr

Songyeon Cho
Samsung Electronics Co.,LTD.(*)
(*) Author(Songyeon Cho)'s contribution was done before
joining Samsung Electronics.

EMail: Songyeon.Cho@gmail.com

Emmanuel Baccelli
Inria

Phone: +33-169-335-511
EMail: Emmanuel.Baccelli@inria.fr
URI: <http://www.emmanuelbaccelli.org/>

TCP Maintenance Working Group
Internet-Draft
Intended status: Experimental
Expires: January 16, 2014

T. Flach
USC
N. Dukkipati
Y. Cheng
B. Raghavan
Google
July 15, 2013

TCP Instant Recovery: Incorporating Forward Error Correction in TCP
draft-flach-tcpm-fec-00.txt

Abstract

Ordinary TCP loss recovery takes at least one round-trip time and as such can increase application-perceived latency, especially for short flows such as Web transactions. TCP Instant Recovery (TCP-IR) is an experimental algorithm that allows a receiving end to recover lost packets without retransmissions, thus potentially saving at least one full round-trip time compared to standard TCP. TCP-IR achieves this by judiciously injecting encoded data segments within a TCP stream. This document describes the TCP-IR algorithm at the sending and receiving ends, along with the required protocol changes.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 16, 2014.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Terminology	3
2. Protocol Overview	3
3. Protocol Details	5
3.1. TCP-IR Option	6
3.2. Negotiation	7
3.3. Encoding Types	8
3.4. TCP-IR Sender	9
3.5. TCP-IR Receiver	9
3.6. Processing Acknowledgements	11
4. Interaction with middleboxes	11
5. Implementation and Performance	12
6. Related Work	13
7. Security Considerations	13
8. IANA Considerations	13
9. References	13
Authors' Addresses	14

1. Introduction

TCP Instant Recovery (TCP-IR) enables a receiver to recover lost data segments instantly without the need for retransmissions from a sender. Standard TCP retransmission-based loss recovery takes at least one RTT for loss detection and recovery.

The main motivation for TCP-IR is to reduce the tail latency of Web transactions. Most Web transfers are short and could finish within a few round-trip times (RTTs), but losses can add multiple RTTs to transfer times and increase the variance in Web page download times. The goal of TCP-IR is to reduce loss detection and recovery to zero RTT while still employing TCP's congestion control principles.

Recovery mechanisms, such as fast recovery and retransmission timeout, are fundamentally RTT dependent. Regardless of how fast network bandwidth grows, the number of RTTs that it takes to recover lost packets does not change. TCP-IR employs forward error correction (FEC) to scale the recovery time inversely with bandwidth and make it independent of RTT. It explicitly trades some network

bandwidth to reduce RTTs for short transfers. Most bandwidth in the Internet is used by large flows such as video, and thus short, latency-sensitive traffic can benefit using a small degree of FEC without hurting bulk flow throughput.

In this document, we specify the TCP-IR mechanism, which requires both sender and receiver changes, to achieve 1-RTT recovery for commonly observed loss scenarios. Instead of complete redundancy for every segment, we employ FEC within TCP. The sender transmits extra FEC segments, which encode previously transmitted segments, so that the receiver can repair a small number of losses. While the use of FEC for transport has been explored in the past, to our knowledge this is the first specification to place FEC within TCP in a way that is incrementally deployable across today's networks with middleboxes.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Protocol Overview

The key idea in TCP-IR is the judicious introduction of a small number of checksum or XOR segments into TCP's data stream such that a receiver can immediately recover lost segment(s) without the need for retransmissions. The core design challenge is in injecting out of band XOR segments within the regular data stream. We outline the main aspects of the protocol below.

Several of the design choices we made are rooted in our measurements and observations of Internet loss patterns as documented in [RECOVERY-SIGCOMM13]. We find that among the flows experiencing losses, most flows lose only one or two consecutive packets, commonly at the tail of a burst. As an example, for bursts of at most 10 packets, ~35% experienced exactly one loss, and an additional 10% experienced exactly two losses. Further, the latter packets for any given burst size are more likely to be lost. Given these findings, we chose a simple XOR-based encoding scheme that can perform instant recovery of a small amount (one or two segments) of packet loss.

A TCP-IR sender and receiver first negotiate the use of instant recovery in the initial handshake. If both hosts support the use of instant recovery, every packet in the connection includes a TCP-IR option.

TCP-IR sender:

1. Periodically in every round-trip time, a TCP-IR sender places the XOR of newly transmitted segments into a single MSS-length checksum packet. The XOR is only computed for new segments not previously included in checksums.
2. Regardless of the sizes of original segments, the sender computes the XORs along MSS byte boundaries. Because every packet carries a payload of at most MSS bytes, such an encoding guarantees that the receiver can instantly recover any single packet loss.
3. The encoded XOR packet uses the same sequence number as the first segment it encodes. The encoded packet carries a flag in the TCP-IR option signaling that the payload is encoded. A receiver uses the flag to disambiguate an encoded packet from a regular (re)transmission, since both segments carry the same sequence number. The option also includes the number of bytes that the payload encodes.

There is no reliability provided for the XOR segments.

TCP-IR receiver:

1. A receiver first establishes if the payload of the received segment is encoded, by checking a flag in the TCP-IR option.
2. Once the receiver establishes that the payload is encoded, it obtains the encoded range of bytes by using the sequence number of the TCP-IR packet and the the number of bytes encoded.
3. The receiver checks for holes in the encoded range. If it received the entire sequence range, the receiver drops the encoded packet. Otherwise, if it is missing at most MSS contiguous bytes, the receiver uses the encoded payload to recover the lost sequence range and forwards it to the regular reception routine, thus allowing 0-RTT recovery.
4. For the purpose of recovering lost segments, a receiver buffers the last fifteen in-order MSS blocks that it ACKed, even if the application layer has already consumed these blocks. Because an encoded packet is the XOR of at most sixteen MSS segments, the receiver can recover any single lost packet by computing the XOR of the encoded payload and the buffered data in the encoding range.
5. If too much data is missing for the encoded packet to recover, the receiver sends a duplicate ACK. This ACK informs the sender that a recovery failed and also denotes the byte ranges lost via the TCP-IR option. The sender marks the byte ranges as lost and triggers a fast retransmit and recovery.

6. TCP-IR does not circumvent congestion control. If the receiver were to simply ACK a recovered packet, it would mask the loss and prevent congestion control during a known loss episode. To perform congestion window reduction upon a successful recovery at the receiver, TCP-IR uses a mechanism similar to explicit congestion notification (ECN). Upon a successful recovery, the receiver enables an R_SUCC flag in the TCP-IR option in each outgoing ACK. The sender in turn triggers a congestion window reduction and sets an R_ACK flag in the TCP-IR option of the next packet sent to the receiver. Once the receiver observes R_ACK in an incoming packet, indicating that the sender reduced the congestion window, it disables R_SUCC for future packets.

Figure 1 gives an example of TCP-IR in action.

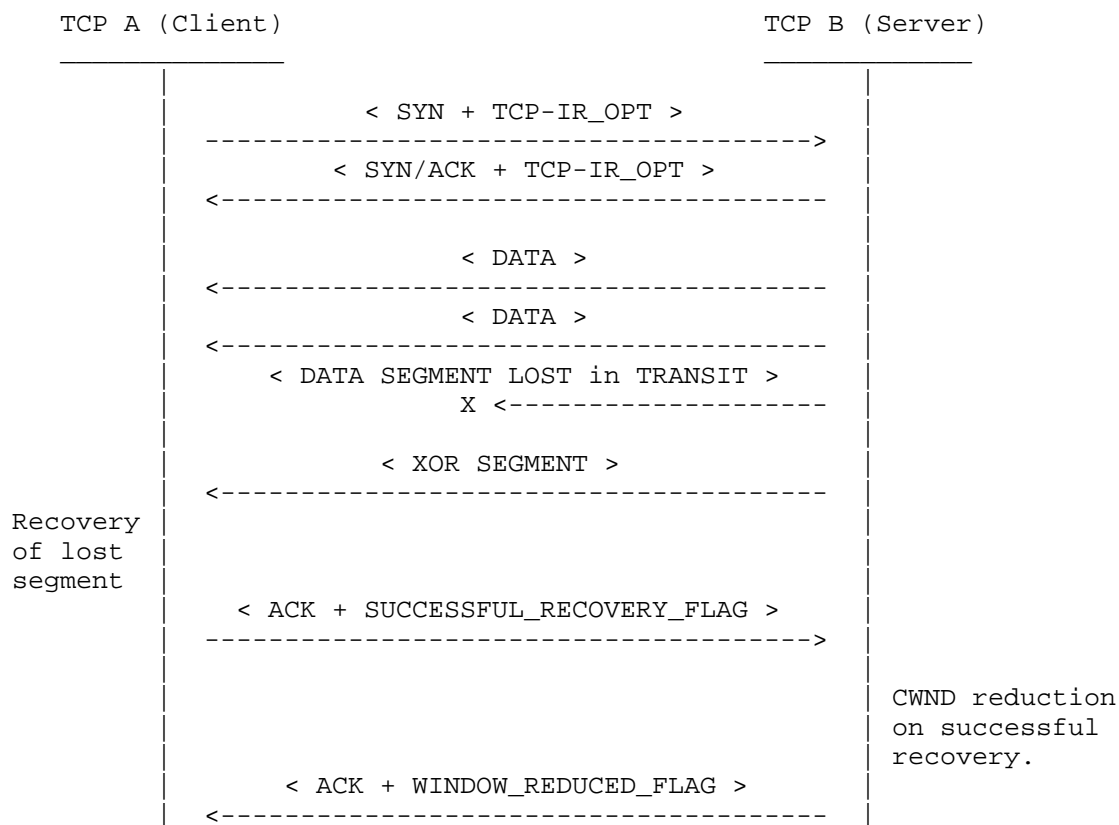


Figure 1: Sample flow using TCP-IR

3. Protocol Details

In the following, we describe the TCP-IR option design, negotiation of instant recovery, the supported encoding schemes, and finally the sender and receiver side algorithms.

3.1. TCP-IR Option

Both the server and the client use a new option to perform the following:

- o Negotiate the use of TCP-IR, including the encoding type.
- o Distinguish encoded packets from regular packets.
- o Communicate the number of encoded bytes in an XOR packet.
- o Acknowledge the recovery of segments and congestion window reductions.
- o Indicate the loss of segments.

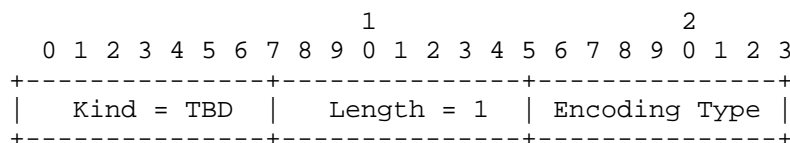


Figure 2: TCP-IR Option format for packets with SYN flag set

During the initial handshake (for packets with the SYN flag set), the option has the format shown in Figure 2. It contains the following fields:

Kind (8 bits)

This MUST be set to the option number for TCP-IR to be determined by IANA.

Length (8 bits)

This MUST be set to the length of the TCP option in octets; its value MUST be 1.

Encoding Type (8 bits)

This SHOULD be set to a value corresponding to a supported encoding type (see Section 3.3).

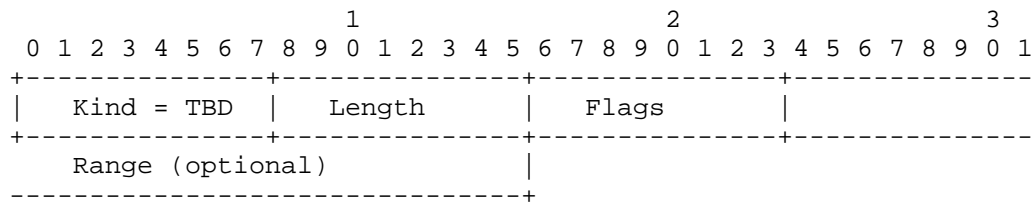


Figure 3: TCP-IR Option format (except for packets with SYN flag set)

For all other packets, the option has the format shown in Figure 3. It contains the following fields:

Kind (8 bits)

This MUST be set to the option number for TCP-IR to be determined by IANA.

Length (8 bits)

This MUST be set to the length of the TCP option in octets; its value MUST be 1, or 4 (if the "Range" field is appended).

Flags (8 bits)

The field can carry the following flags (each represented by one bit):

Bit	Flag Name	Description
0	R_CWR	Congestion Window Reduction Acknowledgement
1	R_SUCCESS	Recovery successful
2	R_FAIL	Recovery failed
3	ENCODED	Packet is encoded
4-7		Unused

The unused bits SHOULD NOT be set.

Range (24 bits, optional)

This field is only used if either the ENCODED or R_FAIL bit in the "Flags" field is set. If the ENCODED bit is set, this field specifies the number of bytes encoded in the payload. If the R_FAIL bit is set, this field specifies the number of bytes considered lost (see Section 3.4 and Section 3.6).

3.2. Negotiation

TCP-IR MUST be explicitly negotiated during the initial handshake. If the negotiation succeeds, both endpoints can send and receive TCP-IR packets. More specifically TCP-IR is enabled if:

1. The receiver sends the SYN packet carrying the TCP-IR option. The encoding type field MUST carry a valid encoding type (see Section 3.3).
2. The sender responds with a SYN/ACK carrying the TCP-IR option. The encoding type field MUST carry the same encoding type as the option in the corresponding SYN packet.
3. All following packets (transmitted by either sender or receiver) MUST carry the TCP-IR option.

If any endpoint receives a packet (after negotiation succeeds) that does not carry the TCP-IR option, the connection MUST be reset. This is necessary because a receiver can no longer distinguish between regular and TCP-IR packets. We recommend tracking these cases to avoid TCP-IR negotiation on future connections.

3.3. Encoding Types

The client can select the encoding type to be used by the TCP-IR module but both endpoints have to support it and agree on it during the initial handshake (see Section 3.2).

Currently the following encoding types are supported and are therefore valid values in the "Encoding Type" field of the TCP-IR option during negotiation:

Value	Type	Description
0	Undefined	
1	Basic XOR	TCP-IR packets carry the XOR of every MSS length segment. One TCP-IR packet encodes up to 16 MSS length segments.
2	Interleaved XOR	TCP-IR packets carry the XOR of every other MSS length segment. One TCP-IR packet encodes up to 8 MSS length segments.
3-255	Undefined	

We selected these encoding types to demonstrate the flexibility of TCP-IR with respect to user preferences (like the acceptable amount of redundancy) and connection properties. Basic XOR can recover a single segment loss of up to MSS bytes in the encoding range. Interleaved XOR enables the recovery of two consecutive segments of up to MSS bytes. This makes Interleaved XOR suitable for connections observing bursty losses, but can double the number of generated TCP-IR packets.

The currently supported encoding types use the MSS value to determine the block size for the encoding process. If the MSS value changes after the initial handshake, TCP-IR MUST be disabled for the remainder of the connection.

3.4. TCP-IR Sender

All packets after the initial handshake carry the TCP-IR option to ensure that the receiver can always distinguish regular packets from encoded packets (packets carrying a payload encoded by the TCP-IR module), or detect the removal of the option by a middlebox. Encoded packets MUST set the ENCODED bit in the "Flags" field of the TCP-IR option; all other packets MUST NOT set the ENCODED bit.

The TCP-IR packet MUST use the same sequence number as the first byte it encodes. This prevents enforcing reliability for encoded packets as well as the overhead of specifying the index of the first encoded byte in a separate option field.

In addition to that, the option in encoded packets MUST carry the "Range" field. The value in the "Range" field specifies the index of the byte after the last encoded byte in the payload relative to the sequence number of the encoded packet.

TCP-IR adds a short delay in the transmission of encoded packets to reduce the probability of losing both the original transmission and the encoded packet in the same loss burst.

The encoding and transmission routine works as follows:

1. Before a regular data packet is forwarded to the IP layer, the TCP-IR timer is armed (unless the timer is already armed). In our prototype implementation the timer is set to a value of $RTT/4$.
2. Once the timer fires, all transmitted segments not encoded before are now encoded according to the negotiated encoding type and the corresponding encoded packets are transmitted immediately. The maximum number of MSS length segments which can be encoded in a single TCP-IR packet depends on the negotiated encoding type (see Section 3.3). As a result, the number of encoded packets created in this step depends on the encoding type and the number of previously un-encoded segments. The option fields in the encoded packet are populated as described in Section 3.1.

3.5. TCP-IR Receiver

Receivers distinguish TCP-IR packets from regular packets by checking the ENCODED bit in the "Flags" field of the TCP-IR option. Encoded

packets are forwarded to the TCP-IR reception routine (described below). If the packet does not carry the TCP-IR option it is discarded and TCP-IR is disabled for the remainder of the connection. To inform the sender that TCP-IR can no longer be used, the receiver sends an acknowledgement without the TCP-IR option.

Additionally, the regular reception routine is modified as follows. The last 15 ACKed MSS length segments remain in the buffer, even if the application layer has already consumed these segments. Segments received out-of-order are already buffered by default and cannot be consumed by the application layer. Since a single TCP-IR packet encodes at most 8 (interleaved XOR) or 16 (basic XOR) MSS length segments, any single segment loss (up to MSS length) in the encoding range can be recovered by the decoder.

The reception routine for TCP-IR packets works as follows:

1. The encoding range of the TCP-IR packet is extracted. As mentioned earlier, the sequence number of the packet specifies the sequence number of the first encoded byte. The sequence number plus the value stored in the "Range" field in the TCP-IR option minus 1 specifies the sequence number of the last encoded byte.
2. If all bytes in the encoding range were already received, skip to Step 5.
3. If lost segments in the encoding range can be recovered (in the case of XOR encoding, a loss of at most one MSS length segment in the encoding range can be handled):
 - a. The lost segments are reconstructed. The matching packet headers are appended to the reconstructed segments and the packets are forwarded to the regular reception routine.
 - b. The R_SUCCESS bit in the "Flags" field of the TCP-IR option is set for all future packets. This includes the (potentially delayed) acknowledgement for the recovered segment. Further details are described in Section 3.6.
4. If none of the segments in the encoding range are recoverable:
 - a. The sequence number of the last byte lost is extracted. The offset between the sequence number of the next expected byte (RCV.NXT) and the last byte lost defines the loss range.
 - b. An acknowledgement is generated with the following requirements for the TCP-IR option.
 - + The R_FAIL bit in the "Flags" field is set.

- + The option carries the "Range" field. The "Range" field encodes the loss range, as described above. The context is maintained, since the acknowledgement number will be set to RCV.NXT.
5. The TCP-IR packet is discarded.

3.6. Processing Acknowledgements

If a receiver instantly recovers losses we want to ensure the sender learns of it so as to not circumvent congestion control [RFC5681]. The R_SUCCESS bit in the "Flags" field of the TCP-IR option informs the sender that the receiver successfully recovered a lost packet. Once the sender observes the R_SUCCESS bit in a packet the following steps are executed:

1. The sender reduces its congestion window.
2. The sender sets the R_CWR bit in the "Flags" field of the TCP-IR option in the next outgoing packet only.
3. The sender does not act on any future observations of the R_SUCCESS bit being set until SND.UNA advances past the SND.NXT value observed at the time when Step 2 was executed. This ensures the congestion window is not reduced multiple times in the same loss episode.
4. Once the receiver observes the R_CWR bit being set in any incoming packet, the R_SUCCESS bit is reset for all future packets.

A failed recovery on the receiver side triggers an explicit acknowledgment sent to the sender to inform it about the segments that are considered lost. This is indicated by the R_FAIL bit being set in the "Flags" field of the TCP-IR option. If the sender observes this bit being set, the following steps are executed:

1. The sender extracts the loss range from the "Range" field in the TCP-IR option. The sequence numbers of the first and last byte lost are defined by the acknowledgement number of the packet, and the acknowledgement number plus the loss range value.
2. The sender marks the appropriate byte range as lost and triggers Fast Retransmit/Recovery.

Explicit notification of loss ranges has the benefit that lost segments are retransmitted faster, avoiding the extra wait time until the RTO fires.

4. Interaction with middleboxes

An important design goal of TCP-IR is compatibility with middleboxes and support for graceful fallback to standard TCP behavior in

situations where middlebox interference prevents proper use of TCP-IR.

Even if hosts negotiate TCP-IR during the initial handshake, it is possible for a middlebox to strip the option from a later packet. To be robust to this, if either host receives a packet without the option, it **MUST** discard the packet and reset the connection. This is necessary since receivers are no longer able to distinguish TCP-IR packets from regular packets.

TCP-IR uses relative sequence numbers to convey metadata (such as the encoding range) between endpoints. This prevents issues in the cases of middleboxes performing sequence number translations.

Some problems caused by middlebox interference (and their solutions in TCP-IR) are not discussed in the current version of this draft:

- o Rewriting of the acknowledgement number if the acknowledged segment was not observed by the middlebox. With TCP-IR this can occur after recovering a lost segment. This issue can be circumvented by retransmitting the recovered segment, even though it is not needed by the other endpoint anymore. This plugs the "sequence hole" in the state of the middlebox.
- o Rewriting payloads of previously seen segments.
- o Packet coalescing and splitting.

5. Implementation and Performance

We implemented TCP-IR in Linux TCP in about 1600 lines of code. 20% of the modular implementation includes the parts common to both the sender and receiver, which are option encoding/decoding, and negotiation during connection setup. 50% of the implementation is the receiver components including detection of an encoded packet, decoding the TCP-IR payload, and generating the right acknowledgements upon a successful or failed recovery. The remainder 30% of the implementation is the sender component which consists mainly of payload encoding and transmission.

We conducted two kinds of experiments. The first was in an emulated setting using loss patterns similar to those observed in our measurement of real traffic. We used the netem module to emulate a 200 ms RTT and both random and correlated loss rates of 2%. TCP-IR reduced the latency for short transfers in lossy environments by 28% in the 90th percentile. The benefits diminish as the minimum number of RTTs necessary to complete the transaction increases (due to the message size) because the time to recover from losses no longer dominates the overall transmission time. TCP-IR is better suited for small transfers common in today's Web.

In the second set of experiments, we used the Web-page-replay tool and dummynet to replay HTTP resource transfers for actual Web page downloads through controlled, emulated network conditions. We tested a variety of popular Web sites, and ran separate tests for Web pages tailored for desktop and mobile clients. As an example, with TCP-IR, the New York Times website takes 15% less time in the 90th percentile until the first objects are rendered on the screen.

Details on performance experiments with TCP-IR are in [RECOVERY-SIGCOMM13].

6. Related Work

Applying FEC to transport, at nearly every layer, is an old idea. [Coding-IEEE2011] suggested placing network coding in TCP, and [CodedTCP-2013] extended this work by implementing a variant over UDP mainly for high loss rate wireless environments. Among others, [AdaptiveFEC-2004] and Tickoo et al. [LT-TCP-2005] explored extending TCP to incorporate FEC. Finally, Maelstrom is an FEC variant for long-range communication between data centers leveraging the benefits of combining and encoding data from multiple sources into a single stream [Maelstrom-2011]. The focus of all of this work is on the performance aspects of using FEC over lossy links. None address the protocol level changes required in TCP to incorporate FEC.

7. Security Considerations

The security considerations outlined in [RFC5681] apply to this document. At this time we did not find any additional security problems with TCP-IR.

8. IANA Considerations

The two Options for TCP-IR used during negotiation and subsequently in every packet of the connection require IANA allocate one value from the TCP option Kind namespace. Experimentation prior to the allocation SHOULD follow [EXPOPT] and use experimental option kind 254 and two magic bytes 0xDC60, and migrate to the new option after the allocation accordingly.

Note to RFC Editor: this section may be removed on publication as an RFC.

9. References

[RECOVERY-SIGCOMM13]

Flach, T., Dukkkipati, N., Terzis, A., Raghavan, B., Cardwell, N., Cheng, Y., Jain, A., Hao, S., Katz-Bassett, E., and R. Govindan, "Reducing Web Latency: the Virtue of Gentle Aggression", Proceedings of the 2013 ACM SIGCOMM , 2013.

[Coding-IEEE2011]

Sundararajan, J., Shah, D., Medard, M., Jakubczak, S., Mitzenmacher, M., and J. Barros, "Network Coding Meets TCP: Theory and Implementation", Proceedings of the IEEE , 2011.

[CodedTCP-2013]

Kim, M., Cloud, J., ParandehGheibi, A., Urbina, L., Fouli, K., Leith, D., and M. Medard, "Network Coded TCP (CTCP)", arXiv:1212.2291. , 2013.

[AdaptiveFEC-2004]

Baldantoni, L., Lundqvist, H., and G. Karlsson, "Adaptive end-to-end FEC for improving TCP performance over wireless links", IEEE Communications Society , 2004.

[LT-TCP-2005]

Tickoo, O., Subramanian, V., Kalyanaraman, S., and K. Ramakrishnan, "LT-TCP: End-to-End Framework to improve TCP Performance over Networks with Lossy Channels ", Proc. of IWQoS , 2005.

[Maelstrom-2011]

Balakrishnan, M., Marian, T., Birman, K., Weatherspoon, H., and L. Ganesh, "Maelstrom: transparent error correction for communication between data centers ", IEEE/ACM Transactions on Networking , 2011.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.

[EXPOPT] Touch, J., "Shared Use of Experimental TCP Options", draft-ietf-tcpm-experimental-options-06 (work in progress), October 2012.

Authors' Addresses

Tobias Flach
University of Southern California
941 Bloom Walk
Los Angeles, California 90089
USA

Email: flach@usc.edu
URI: <http://nsl.cs.usc.edu/~tobiasflach>

Nandita Dukkhipati
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, California 94043
USA

Email: nanditad@google.com

Yuchung Cheng
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, California 94043
USA

Email: ycheng@google.com

Barath Raghavan
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, California 94043
USA

Email: barath@google.com