Network Coding Meets TCP

Michael Mitzenmacher

Joint work with

Jay-Kumar Sudararajan, Devavrat Shah,

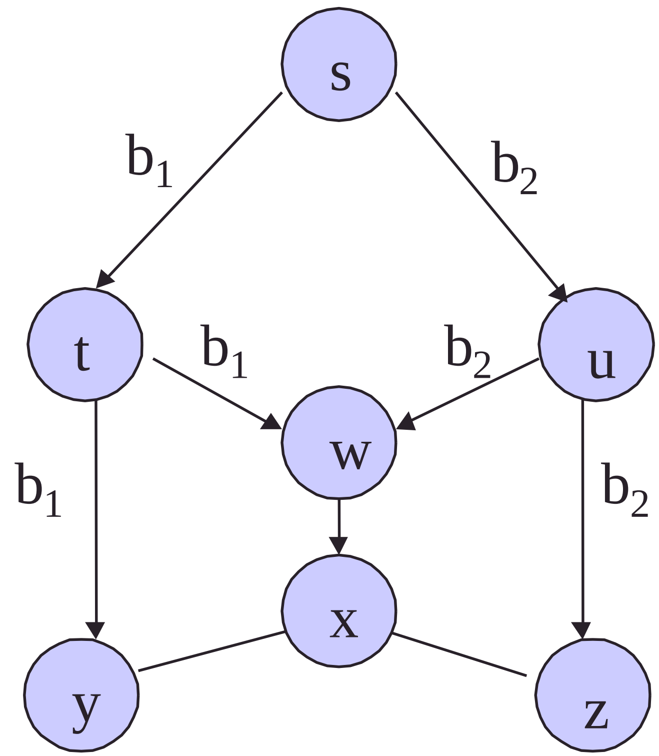
Muriel Medard, Joao Barros

Network Coding

- Packets can be encoded arbitrarily, not just by end nodes, but also by nodes within the network.
  – End-to-end codes a special case.
- Standard example : butterfly network.

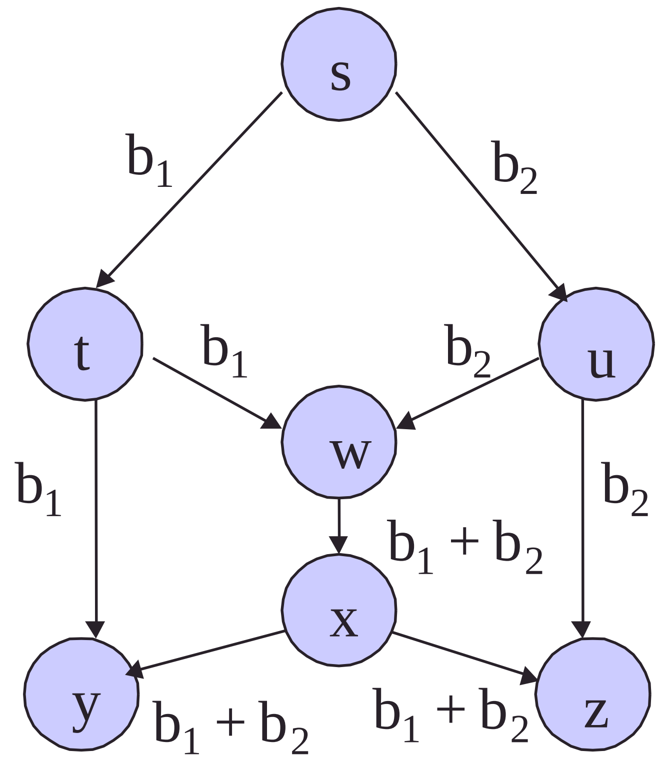
Butterfly Example

- Want both bits to get to both y and z as quick as possible.
  - Delay, throughput.
- Bottleneck at link from w to x.



Butterfly Example

- Want both bits to get to both y and z as quick as possible.
  - Delay, throughput.
- Bottleneck at link from w to x.
- Solution : encode by sending linear combination of bits.



Practice?

- Will network coding achieve wide use in practice, or just a mathematical toy?
  - Jury is still out… but lots of believers.
    - Lots of theory, projects.
    - Avalanche, COPE, MORE,…
- Potential problem: incremental deployment / backward compatibility.
  - Standard problem for anything new.

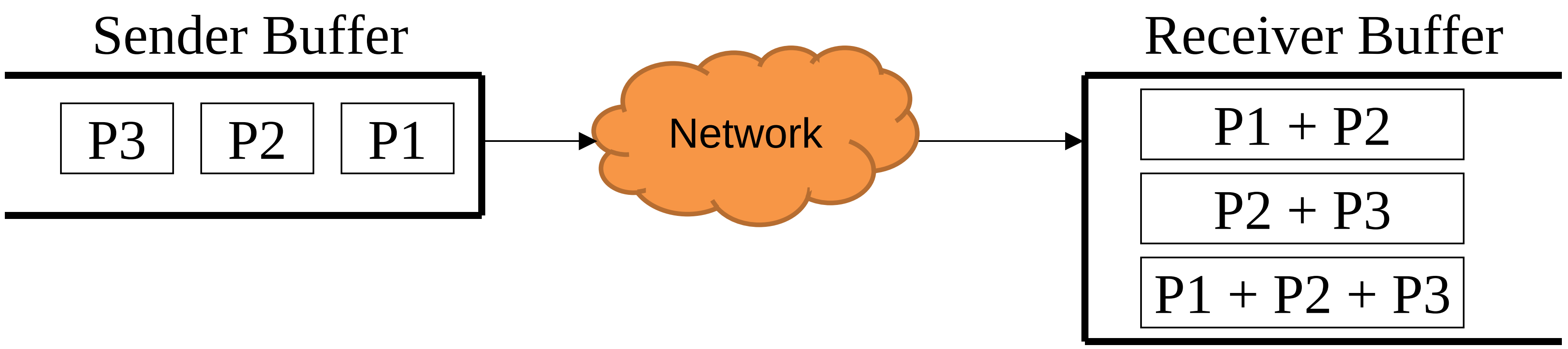
TCP and Coding

- For incremental deployment, best to be compatible or friendly with TCP.
- Not easy; TCP not designed for coding.
- TCP *combines* reliability and congestion control; with coding, you don't want reliability.
  - But still the need for congestion control.

Comparison : Fountain Codes

- Fountain codes use coding *just at endpoints*.
  - Random XORs of packets.
- Congestion control issues a big problem for usage. TCP-friendliness/TCP-compatibility.
- Special schemes designed for:
  - Multicast congestion control.
  - Long-distance, high-bandwidth connections.

The Problem



Can't acknowledge a packet until you can decode.
Usually, decoding requires a number of packets.
Code / acknowledge over small blocks to avoid
delay, manage complexity.

Compare to ARQ

Context: Reliable communication over a (wireless) network of packet erasure channels

ARQ

- Retransmit lost packets
- Low delay, queue size
- Streaming, not blocks
- Not efficient on broadcast links
- Link-by-link ARQ does not achieve network multicast capacity.

Network Coding

- Transmit linear combinations of packets
- Achieves min-cut multicast capacity
- Extends to broadcast links
- Congestion control requires feedback
- Decoding delay: block-based

Goals

- Devise a system that behaves as close to TCP as possible, while masking non-congestion wireless losses from congestion control where possible.
  - Standard TCP/wireless problem.
- Stream-based, not block-based.
- Low delay.
- Focus on wireless setting.
  - Where network coding can offer biggest benefits.
  - Not necessarily a universal solution.

Main Idea : Coding ACKs

- What does it mean to “see” a packet?
- Standard notion: we have a copy of the packet.
  - Doesn’t work well in coding setting.
  - Implies must decode to see a packet.
- New definition: we have a packet that will allow us to decode *once enough* useful packets arrive.
  - Packet is useful if linearly independent.
  - When enough useful packets arrive can decode.

Coding ACKs

- For a message of size *n*, need *n* useful packets.
- Each coded packet corresponds to a degree of freedom.
- *Instead of acknowledging individual packets, acknowledge newly arrived degrees of freedom.*

Coding ACKs

Original message : $p_1, p_2, p_3\ldots$

Coded Packets

$4p_1 + 2p_2 + 5p_3$

c_1	4	2	5	0	0	0	0
c_2	3	1	2	5	0	0	0
c_3	1	2	3	4	1	0	0
c_4	3	3	1	2	1	0	0
c_5	1	2	5	4	5	0	0

$$\begin{pmatrix} 4 & 2 & 5 & 0 & 0 & 0 & 0 \\ 3 & 1 & 2 & 5 & 0 & 0 & 0 \\ 1 & 2 & 3 & 4 & 1 & 0 & 0 \\ 3 & 3 & 1 & 2 & 1 & 0 & 0 \\ 1 & 2 & 5 & 4 & 5 & 0 & 0 \end{pmatrix}$$

Coding ACKs

Original message : $p_1, p_2, p_3\ldots$

Coded Packets

$4p_1 + 2p_2 + 5p_3$

c_1	4 2 5 0 0 0 0
c_2	3 1 2 5 0 0 0
c_3	1 2 3 4 1 0 0
c_4	3 3 1 2 1 0 0
c_5	1 2 5 4 5 0 0

$$\begin{pmatrix} 4 & 2 & 5 & 0 & 0 & 0 & 0 \\ 3 & 1 & 2 & 5 & 0 & 0 & 0 \\ 1 & 2 & 3 & 4 & 1 & 0 & 0 \\ 3 & 3 & 1 & 2 & 1 & 0 & 0 \\ 1 & 2 & 5 & 4 & 5 & 0 & 0 \end{pmatrix}$$

When c_1 comes in, you've "seen" packet 1; eventually you'll be able to decode it. And so on…

Coding ACKs

Original message : $p_1, p_2, p_3\ldots$

Coded Packets

$4p_1 + 2p_2 + 5p_3$

c_1	4	2	5	0	0	0	0
c_2	3	1	2	5	0	0	0
c_3	1	2	3	4	1	0	0
c_4	3	3	1	2	1	0	0
c_5	1	2	5	4	5	0	0

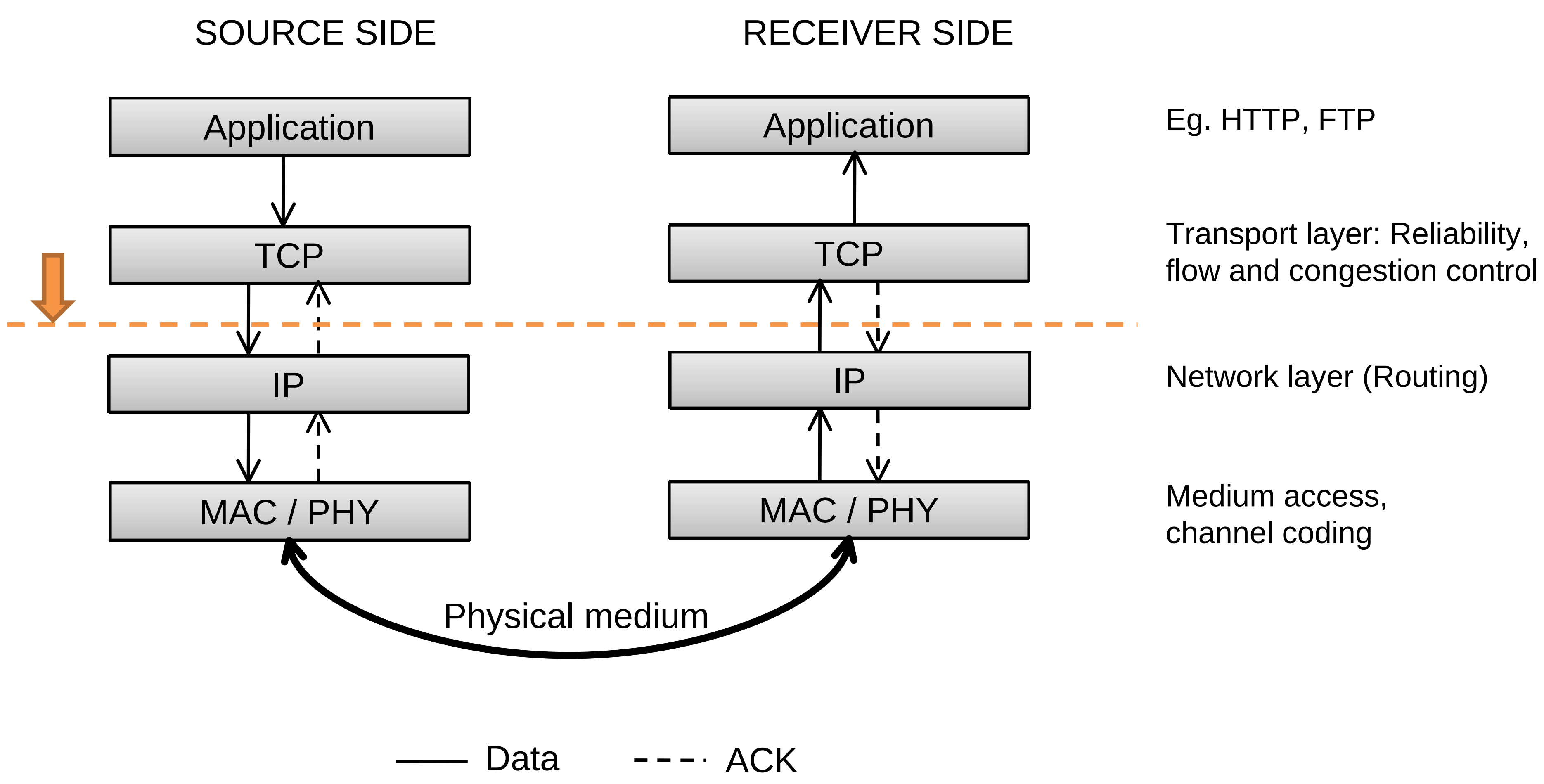
$$\begin{pmatrix} 1 & 4 & 5 & 3 & 0 & 0 & 0 \\ 0 & 1 & 3 & 2 & 6 & 0 & 0 \\ 0 & 0 & 1 & 6 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Use Gaussian elimination as packets arrive to check for a new seen packet.

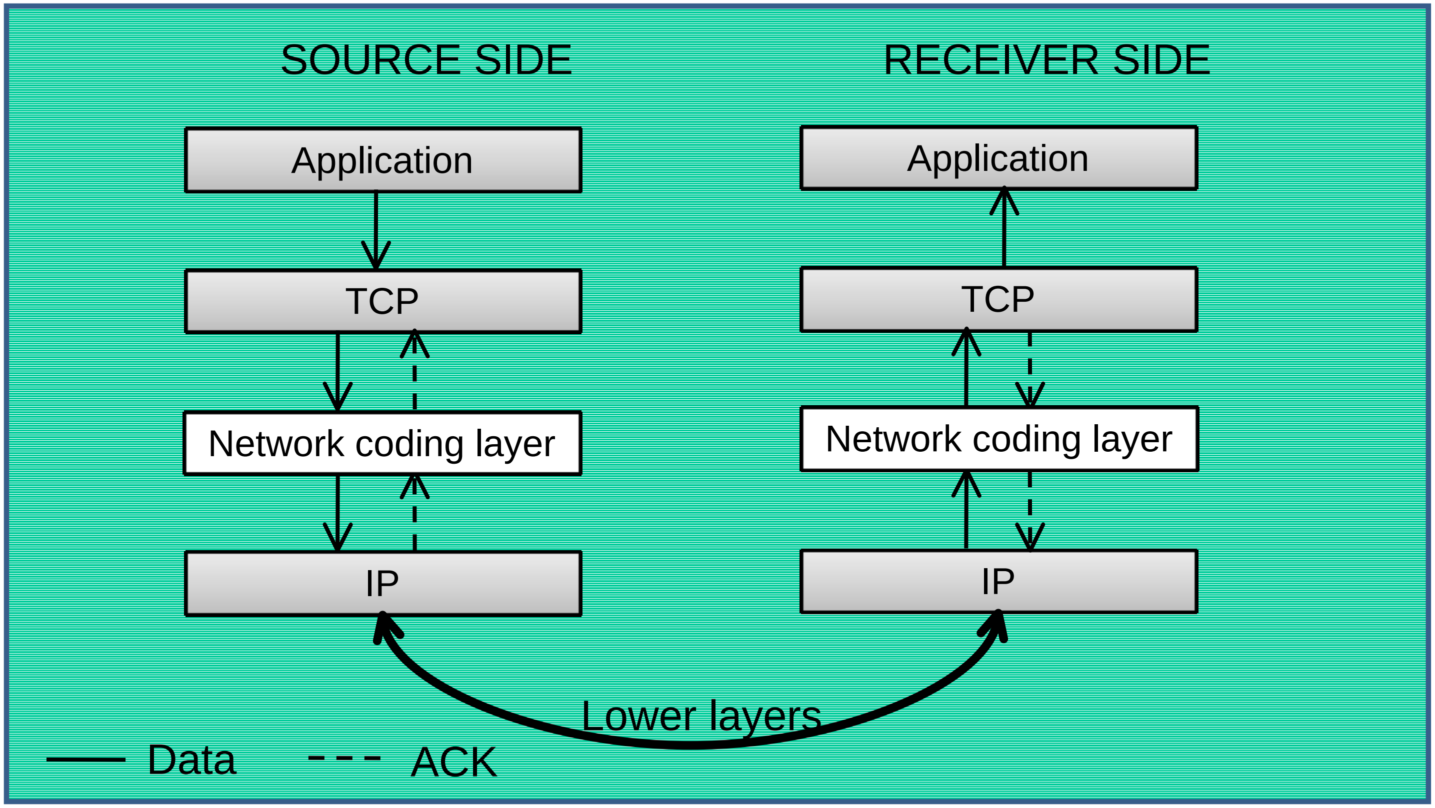
Formal Definition

- A node has *seen* a packet p_k if it can compute a linear combination p_k+q where q is a linear combination of packets with index larger than k.
- When all packets have been seen, decoding is possible.

Layered Architecture



TCP using Network Coding



The Sender Module

- Buffers packets in the current window from the TCP source, sends linear combinations.
- Need for redundancy factor *R*.
  - Sending rate should account for loss rate.
  - Send a constant factor more packets.
  - Open issue : determine *R* dynamically?

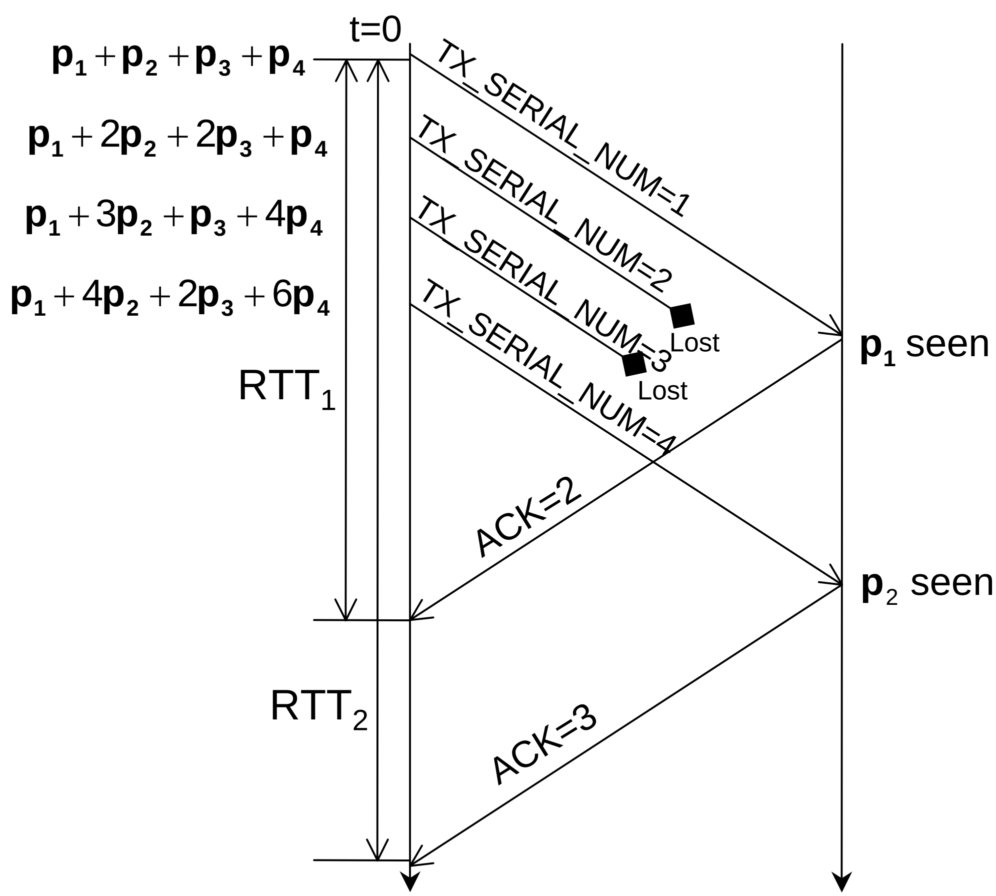
Redundancy

- Too low *R*
  - TCP times out and backs off drastically.
- Too high *R*
  - Losses recovered – TCP window advances smoothly.
  - Throughput reduced due to low code rate.
  - Congestion increases.
- Right *R* is $1/(1-p)$, where p is the loss rate.

Which TCP to Use?

- Use redundancy to match sending rate to desired data rate.
  - Masking wireless losses not due to congestion.
  - TCP Reno reacts to losses; does not seem suitable here.
    - Continuing work – make this approach TCP Reno compatible.
- Instead use TCP Vegas.
  - Sets window based on Round Trip Times.
  - We use RTTs not of packets, but of degrees of freedom.

Measurement of RTTs



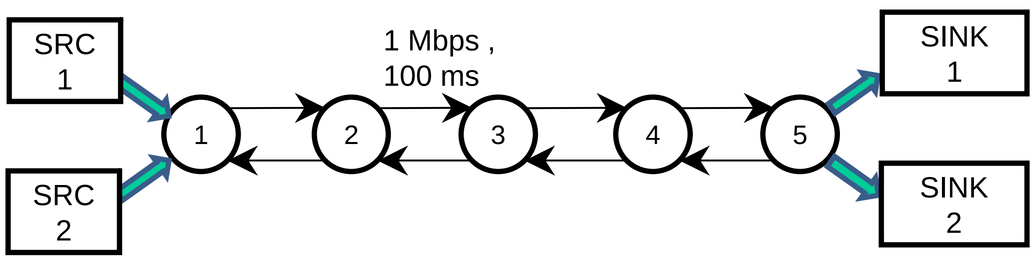
The Receiver Module

- Acknowledgment: ACK a packet upon seeing it (even before it is decoded).
- With high probability (if field size is large), every random linear combination will cause *next unseen* packet to be seen.
- Buffer incoming linear combinations until they can be decoded.
  - Possibly can decode early.
  - Interesting design tradeoff for future work.
- Upon decoding, deliver the packets to the TCP sink.

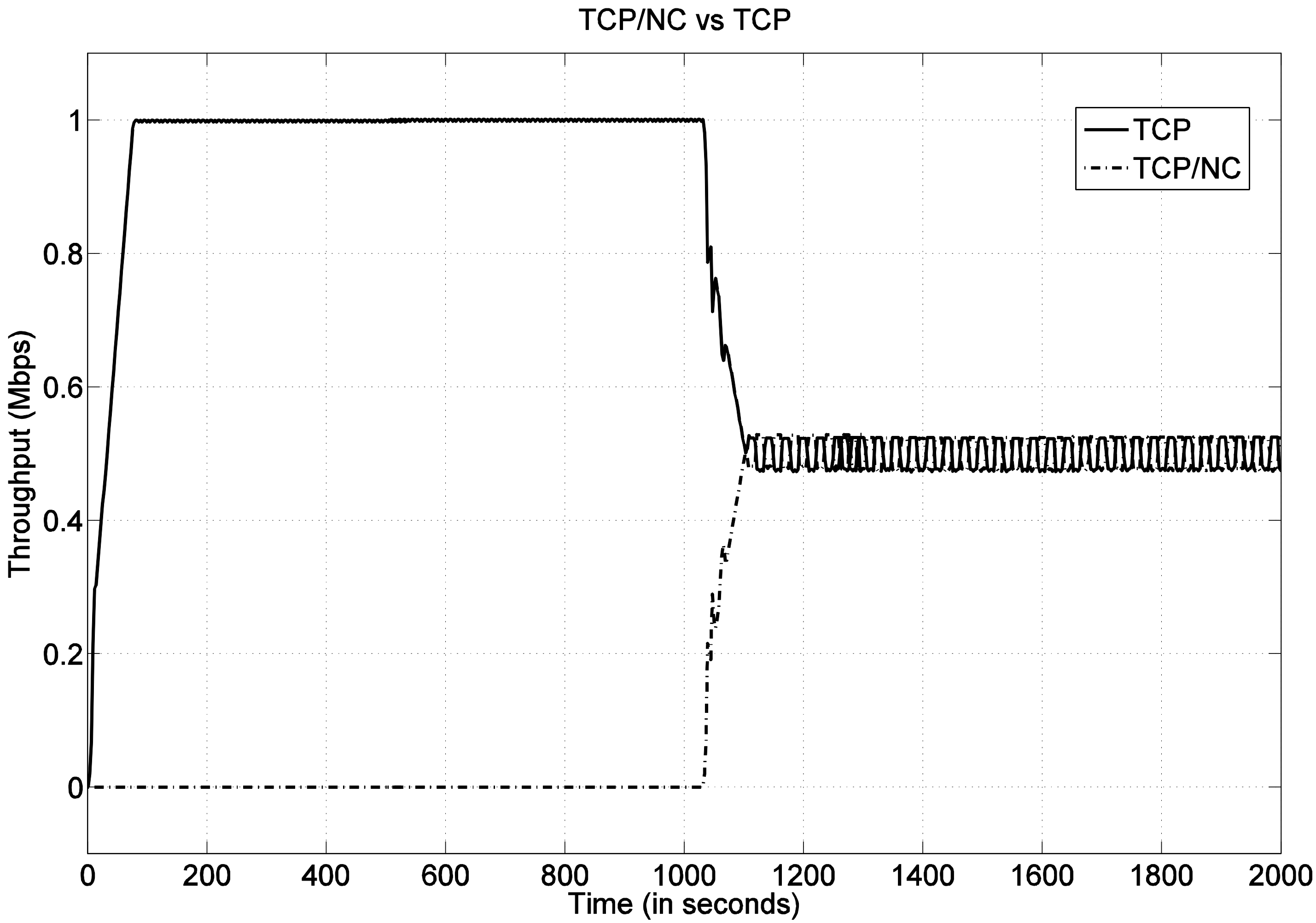
Decoding Early

$$\begin{pmatrix} 4 & 2 & 5 & 0 & 0 & 0 & 0 \\ 3 & 1 & 2 & 5 & 0 & 0 & 0 \\ 1 & 2 & 3 & 4 & 1 & 0 & 0 \\ 3 & 3 & 1 & 2 & 1 & 0 & 0 \\ 1 & 2 & 5 & 4 & 5 & 0 & 0 \end{pmatrix}$$

Some Simulations

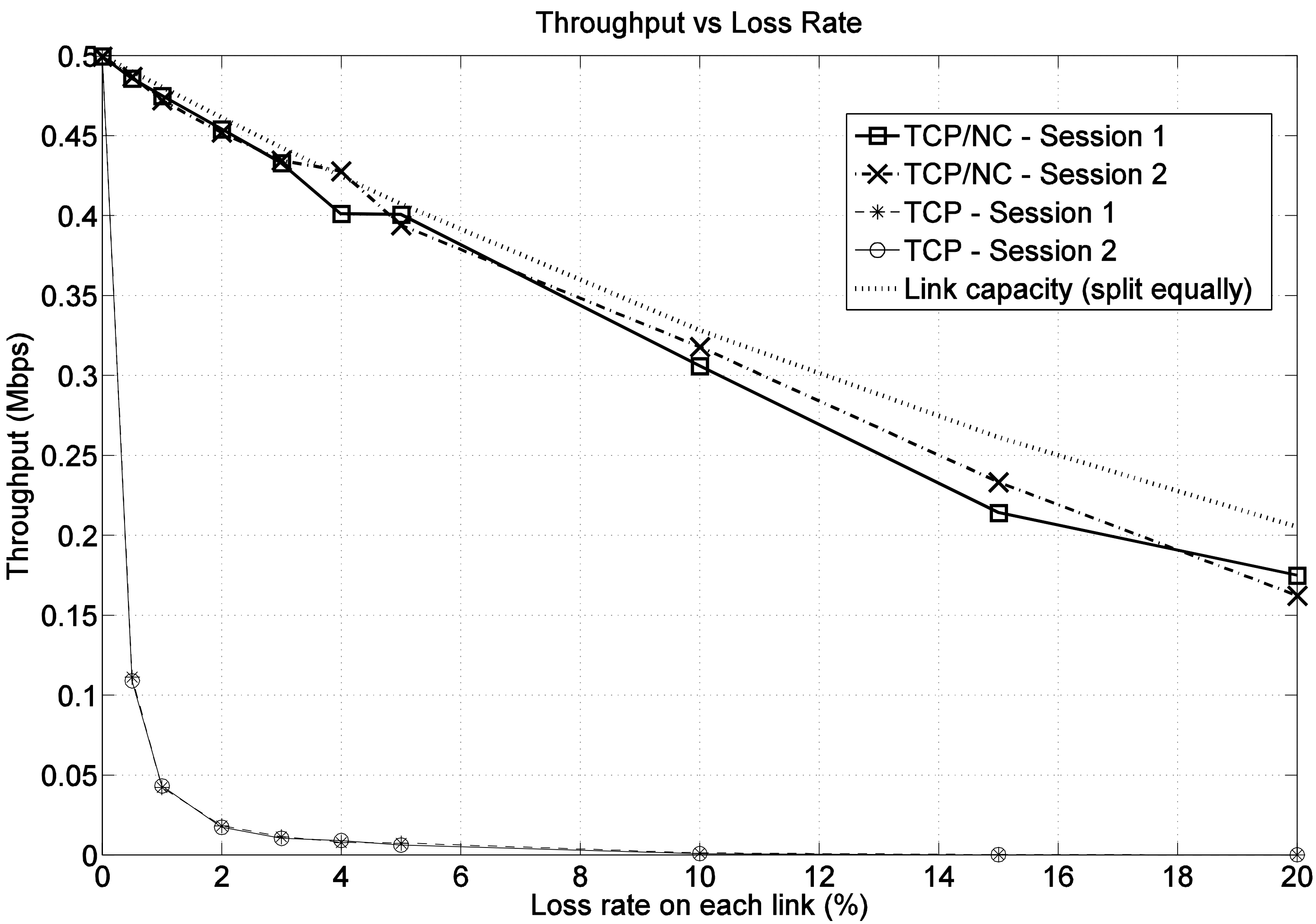


Fairness



0% Loss Rate, Redundancy 1

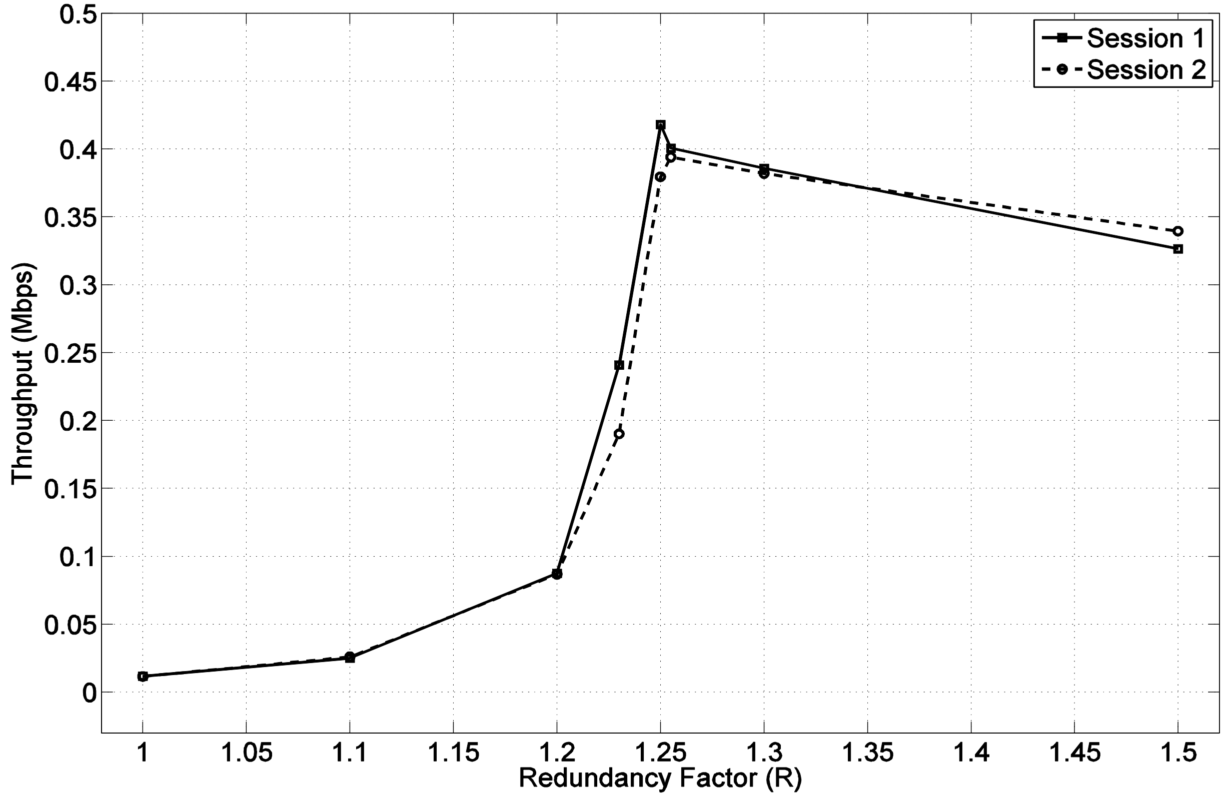
Resilience to Losses



Caveats

- Does not use link layer retransmission.
  - Would help TCP under high loss rates!
- Network coding headers.
  - Need to give coefficients for linear combination!
  - Shared pseudorandom generators help.
- Assumes large field size.
  - Small field size might lead to non-useful packets.
  - In practice, field size of 256 (8 bits) very effective.
- Decoding time.

Redundancy factor



Overall loss rate is roughly 20%

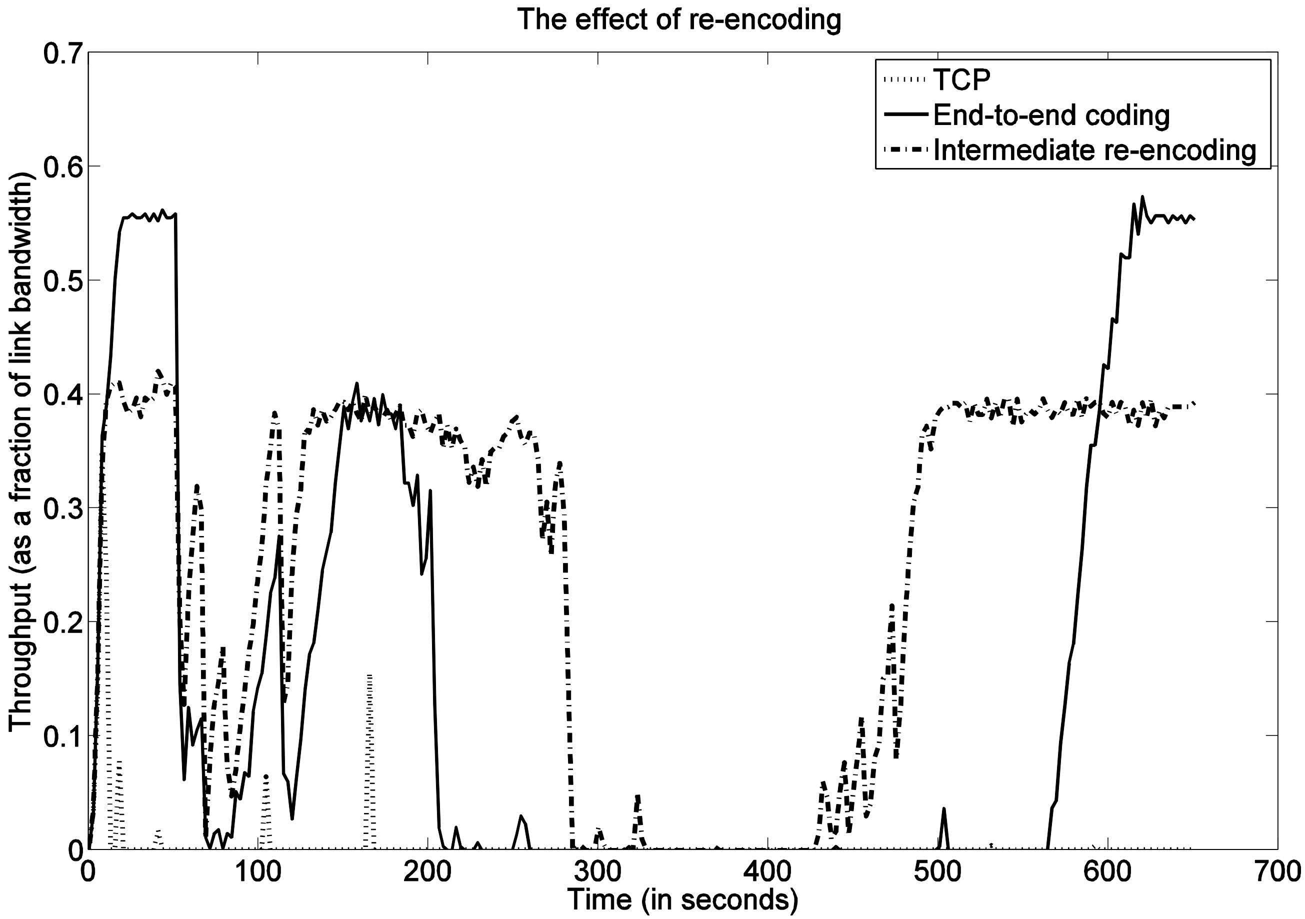
Redundancy Behavior

- Overshooting optimal redundancy : graceful slowdown of throughput.
- Undershooting : less graceful.
  - TCP timeouts.
- But even $R = 1$ is better (by approx. factor of 2) over unmodified TCP.

Re-encoding Experiment

- To see if true network coding (not just end-to-end) is helpful.
- 4 node network, losses along all link.
  - But biggest losses on last link.
- Re-encode along last link.
  - Node has a buffer, sends linear combinations of buffered packets.
  - R for sender is 1.8, for node 3 is 1.5.

Re-encoding



TCP : 0.0042 Mbps ; Coding E-to-E : 0.1420 Mbps ;
Re-encoding : 0.2448 Mbps

Conclusions

- New coding layer proposed between TCP and IP.
- Novel ACK mechanism provides clean interface between network coding and existing congestion control protocols.
- Ideas also work with intermediate node coding.
- Possible extensions to multipath TCP and to multicast sessions.
- Not a final solution, but a step towards realizing the potential of network coding in practice.
  - Proof of concept ; theory.
  - Next stage: deployments underway.

Other Recent Work of Interest

- Hash-Based Techniques for High-Speed Packet Processing
  - A. Kirsch, M. Mitzenmacher, and G. Varghese
  - Survey article
- Why Simple Hash Functions Work: Exploiting the Entropy in a Data Stream
  - M. Mitzenmacher and S. Vadhan
  - Explains why simple hash functions work so well for hash tables, Bloom filters, etc.
  - Randomness in data “combines” with randomness in choice of hash function.

More About Me

- Website: www .eecs.harvard.edu/~michaelm
  - – Links to papers
  - – Link to book
  - – Link to blog : mybiasedcoin
    - mybiasedcoin.blogspot.com