Wireless (Internet) Routing

Opportunistic Routing, Network Coding, TCP over Wireless
Outline

- Review of last time
- Opportunistic Routing
  - What is opportunistic routing?
  - ExOr
  - SOAR
  - Can we use opportunistic transmission paradigm elsewhere?
- TCP on Wireless Networks
Review of Last Time

- Basics of routing in wireless multi-hop networks
  - Proactive vs. reactive
  - DSR, DSDV, AODV

- Link metrics
  - ETX, ETT, WC-ETT
  - Estimators
Why Opportunistic Routing? To take Advantage of Diversity

- Wireless is a broadcast transmission medium
  - Can we turn interference from foe to friend
- Diversity = More than one path: a fundamental paradigm to cope with wireless transmission medium
  - MIMO: multiple antennas
  - Opportunistic scheduling/transmission
The Benefits of Diversity: Two Examples

Take advantage of short or long transmissions
Best case: independent loss events
Example 1: Total ETX for Various Loss Probability and Number of Nodes
Opportunistic Routing: An Example
Challenges of Opportunistic Routing

- Broadcast transmission: how to ensure reliability?
  - Why?
  - How to recover from losses

- Forwarder selection: how to select the “right” set of forwarders with minimum overhead
  - What is the best set of forwarders?
  - What is the optimal number of forwarders?
  - What is the right metric for forwarder selection?
  - How to avoid duplicate transmissions?
  - How to minimize duplicate transmissions?
  - How to minimize the cost of coordination?
ExOR: Opportunistic Multi-Hop Routing for Wireless Networks

- Sigcomm 2005: http://dx.doi.org/10.1145/1080091.1080108

- Objective: take advantage of diversity gain with off-the-shelf hardware (i.e. 802.11)

- Key challenge: how to ensure the best receiver forwards the packet?
ExOR: Key Design Decisions

- Packet batches
  - Nodes send batches of packets
  - Operate per batch
  - To reduce overhead of coordination

- Sender includes ordered/prioritized list of forwarders
  - Priority: closeness of destination (need metric)
  - Forwarders send in order, but only packets not yet acknowledged

- 90/10 tradeoff: per batch, first 90% ExOR, last 10% traditional routing
**ExOR: Node State**

- **Per batch, node keeps**
  - Packet buffer: to store successfully received packets
  - Local forwarder list: copied from a packet of packet buffer
  - Forwarding timer: when to start sending packets from the packet buffer, is adaptive
  - Transmission tracker: record transmission rate of currently sending node + remaining packets, used to update the forwarding timer
  - Batch map: for each packet in a batch, store the highest priority node who received the packet

- **Fragments vs. batches**
  - Fragment = set of packets in packet buffer of sender

---

**Packet header format**

- **Size of the current fragment < Batch size**
- Packet offset within fragment

---

**Current sender offset in FwdList**
ExOR: Mode of Operation

1. Batch preparation
   - Select unique batch ID, select fwd. list, broadcast each packet in batch

2. Forwarder list selection
   - Priority: distance to the destination using ETX along shortest path
   - Select only nodes which would transmit at least 10% of the total transmissions in batch

3. Packet reception
   - If node not in fwd. list, drop packet
   - Replace node ID in batch map if source has higher priority than current batch map
ExOR: Forwarder List Selection
ExOR: Mode of Operation

4. Scheduling of transmission
   - Goal: schedule batch transmission time to (1) allow higher priority nodes send first (2) have only one node send at time
   - Forwarding timer: continuously updated to estimate end of previous fragment
     - Set to current sender’s rate times number of packets remaining to be sent (in fragment)
     - Sender’s rate is estimated by transmission tracker
   - If no information: assumes 5 packet transmissions per higher priority node
   - Node sends only packet not yet received by higher priority node
   - Destination sends ten batch map when its turn comes

5. Completion
   - Node stops sending when over 90% of packets in batch have been received
   - Destination requests missing packet from source using traditional routing
**ExOR: Example 1**

Blue: batch of packet

Green: batch map
ExOR: Example 1

- Src sends 1 and 2
- Fragment at N1 is 1 and 2
ExOR: Example 1

- Src sends 1, 2, 3 and 4
- 3 received at N1 and N2, 4 received at N2 (unknown to N1)
ExOR: Example 1

- N2 sends 3 and 4
- N1 updates batch map: sets N2 for packet 3 and 4
ExOR: Example 1

- N1 sends 1 and 2 only: 3 and 4 have already been sent by higher priority node
ExOR: Example 2

- What is the fragment size and fragment numbers at N2?
  - Size 2, fragment number for packet 3 is 1 and for packet 4 is 2
N1 receives packet 3 from N2: there is still one packet N2 has to send
- N1 can estimate the time to forwarding its fragment
- What is N1 fragment size and numbers?

Fragment size at N1 is 3. Offset is 1,2,3 for packets 1,2,3
ExOR: Performance Overview

- 802.11b roofnet testbed, 1 Mbit/s
- 65 node pairs selected randomly
- Throughput obtained with transmissions of a 1 Mbyte file

Batch size: 100
Median throughputs: 240Kbit/s for ExOR
121Kbit/s for Traditional
ExOR: Discussion

- Need full topology knowledge: see forwarder list
  - Forwarder node selection not robust in sparse network
  - Forwarder list can be large
  - What if topology changes during batch transmission?
  - Proactive topology collection

- Poor spatial reuse
  - Only one node can send at a time

- Forwarding path can diverge

- What about delay?

- Not TCP compatible: a proxy is used
  - Content based protocol: rate control is implicit
  - ExOR mechanism would interact badly with TCP congestion control mechanisms and ack transmissions

- How to support multiple flows/transmissions?
SOAR: Simple Opportunistic Adaptive Routing Protocol for Wireless Mesh Networks

- IEEE TMC 2010:
  http://dx.doi.org/10.1109/TMC.2009.82

- Objective: take advantage of diversity gain with off-the-shelf hardware (i.e. 802.11)
  - Flow-based protocol instead of content-based

- Key Challenges:
  - Rate-control to avoid congestion at preliminary hops (even with TCP)
  - Maintaining low overhead while operating on a per-packet basis
SOAR: Keys Design Decisions

- SOAR operates on a packet basis

SOAR comprises four main components

1. Forwarding candidates selection
   - Design goal: avoid diverging “paths”, minimize duplicates

2. Priority timer-based forwarding
   - To enable the best node to forward the packet, minimize collisions

3. Local loss recovery

4. Rate control
   - Adapt source transmission rate to network conditions
SOAR: Mode of Operation

- Forwarding candidate selections
  - Goal: find good trade-off between progress and reliability
  - Design decision: try to stay close to the shortest-path
  - Fwd. candidates contain: best next-hop + selection
  - Initial selection: 4 conditions
    - C1: ETX of fwd. candidate smaller than ETX of sender
    - C2: Distance between fwd. candidate and sender smaller than threshold, threshold should be adaptive, depends on link to best next-hop
    - C3: Distance between fwd. candidate and next-hop smaller than threshold
    - C4: Distance between any fwd. candidate is within a threshold
  - Pruning: at most N nodes, but less if loss of virtual link is below a threshold
SOAR: Mode of Operation

- Priority timer-based forwarding
  - If node not in fwd. list, drop packet. When to forward?
  - i-th node on the fwd. list sets timer to \((i - 1) \times \delta\)
  - Overhearing or ACK cancel the timer

- Local loss recovery
  - Goal: best-effort reliability
  - Design decision: per-hop network layer ACK
  - Piggyback ACK on data packet + stand-alone ACK
  - ACK format
    - Cumulative/selective ACK: 32 bytes bitmap + start seq. number
    - Cross-flow ACK
      - Add cross-flow ACK only if packet was received
  - Retransmission: simple RTO estimator using non-retransmitted packets
SOAR: Mode of Operation

Rate Control

- Observation: (1) TCP congestion control at the transport layer is not enough for wireless multi-hop networks (2) UDP needs to be rate-controlled
- Design decision: infrequent but regular end-to-end ACK
- ACK format: same as per-hop ACK + total received number of unique packet
- Transmission rate control uses simple window-based protocol
  - Set window size to min. value if number of acked packets is small than min. value
  - Increase window by $incr$ otherwise
SOAR: Performance Overview

- IEEE 802.11a, 6 Mbit/s, CBR traffic at 6 Mbit/s, 1-minute transfer, 30 runs
SOAR: Discussion

- Need full topology knowledge
  - Proactive topology collection

- Most likely not TCP compatible
  - SOAR rate control and block ack might interfere with TCP congestion control
  - SOAR + reliability mechanism could replace TCP in a wireless network
Discussion on Opportunistic Routing

- Current implementation use 802.11 in broadcast mode
  - Rate adaptation algorithms? Tradeoff between number of forwarders and transmission rate
- Topology collection
- Reliability
- TCP support?
  - Jitter, reordering
- No TCP?
  - Multiple flow fairness and congestion control
  - Reliability
  - End-to-end support
Network Coding for Wireless Networking: a Short Primer

- What is network coding?
- A simple example (how many transmissions?)

```
A -----> S -----> B
  |       |
a    ^    b
  |
A ----< S ----< B
  |       |
b    ^    a
  |
A ----< S ----< B
  |       |  
a    ^    a
  |
```

"Network Coding: An Instant Primer", http://dx.doi.org/10.1145/1111322.1111337
Network Coding for Wireless Networks: Can we do Better?

- Use XOR, why?

\[
\begin{array}{ccc}
A & \rightarrow & S & \rightarrow & B \\
A & \rightarrow & S & \leftarrow & B \\
A & \leftarrow & S & \rightarrow & B \\
A & \leftarrow & S & \leftarrow & B \\
\end{array}
\]

<table>
<thead>
<tr>
<th>XOR table</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0</td>
</tr>
<tr>
<td>0 1 1 1</td>
</tr>
<tr>
<td>1 0 1 1</td>
</tr>
<tr>
<td>1 1 0 0</td>
</tr>
</tbody>
</table>

Operations in GF(2)

<table>
<thead>
<tr>
<th>+</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[a \text{ XOR } b\]
Network Coding for Wireless Networks: Example in Detail

- Packet a = 0101
- Packet b = 1101

- Q1: a XOR b?

- a XOR b = 1000 = c

- At A: c XOR a = 1000 XOR 0101 = 1101 = b
- At B: c XOR b = 1000 XOR 1101 = 0101 = a
Discussions

- Q2: What knowledge is assumed at A and B?
- Computational overhead?
Network Coding for Wireless Networks: Linear Network Coding

- Assume packets have L bits
  - Q3: What if two packets don’t have the same size?

- S consecutive bits are symbols over the field $F_{2^S}$
  - Operations on strings of S bits
  - Typically: $S=2$, or $S=8$ (operations on bytes)
  - Addition, subtraction, multiplication and division operations are defined over the numbers $0, 1, ..., 2^S-1$ (with $S=8$, 0 to 255)
  - Q4: why $S=8$?

- Linear combinations
  - Is not concatenation
  - Addition and multiplication in $F_{2^S}$
  - Q5: What is the size of a linear combination of packets of L bits?
  - Q6: Why linear?
Encoding: Linear Combination

- Assume $n$ original packets: $P_1, P_2$ to $P_n$
- Pick $n$ coefficients
  - $g_1, g_2, ..., g_n$
  - *Encoding vector* is $g = (g_1, g_2, ..., g_n)$

- Encoding:
  \[ Z = \sum_{k=1}^{n} g_k P^k \]
  - The summation occurs for every symbol position:
    \[ Z_i = \sum_{k=1}^{n} g_k P_i^k \]
  - $Z$ is the *information vector*

- Encoded packet comprises the $Z$ and $g$ i.e $(g, Z)$
Encoding: it Can be Recursive (Forwarding)

- Set of $m$ encoded packet at a node
  - $(g^1, X^1), (g^2, X^2), \ldots, (g^m, X^m)$

- Node pick new coefficients $h = (h_1, h_2, \ldots, h_n)$
- Node generate new encoded packet $(g', X')$

$$X' = \sum_{k=1}^{n} h_k X^k$$

- Achtung, coefficients are with respect the original packet

$$g'_i = \sum_{k=1}^{m} h_k g^k_i$$
Decoding

- Basic idea: solving a linear system of equations
- Assume node received \((g^1, X^1), (g^2, X^2), \ldots, (g^m, X^m)\)
  - Remember, we had \(n\) original packets
  - We need \(m \geq n\)
- Q7: is \(m \geq n\) sufficient? Why?

\[
\left\{ X^j = \sum_{i=1}^{n} g_i^j M^i \right\}, j = 1, K, m
\]

- Receiver puts \((g^k, X^k)\) row-by-row to build the decoding matrix
- Gaussian elimination (works great on finite fields)
How to Select the Linear Combinations?

- **Equivalent:** how should a node choose the coefficients $g_1, g_2, \ldots, g_n$

- **Simple:** each node should choose the coefficients in a completely independent and decentralized fashion
  - Random network coding
  - Probability of choosing linearly dependent combinations is related to the field size $2^s$
  - Close to negligible for $S=8$

- **Alternative:** deterministic algorithms to design network codes
  - Decentralized is feasible in some network configurations
Finite Field Operation with S=8

- Discrete logarithms
- Use the generator $\alpha$
- Any non-zero element $x$ can be written as

$$x = \alpha^{\log(x)}$$

  - $\log(x)$ is the logarithm
  - $\log(xy) = \log(x) + \log(y)$

- Multiplication and division implemented using table lookup on $x$ to $\log(x)$ (and vice-versa)

- Valid for small $S$
  - In general: map sequence of $S$ bit to polynomial
Applications of network coding

- P2P file distribution: Microsoft Avalanche
- Wireless networks
  - Bidirectional traffic
  - Mesh network
  - Many-to-many broadcast
  - Data-gathering in sensor networks
- Network tomography
- Network security
- On-chip communication
Q&A

- Q1: apply the XOR operation element-wise
- Q2: they need to know that A was XORed with B
- Q3: zero-padding
- Q4: operations on bytes can be efficiently implemented
- Q5: L
- Q6: Because coding and decoding is easier
- Q7: no, because some of the combinations might not be linearly dependent
XORs in the Air: Practical Wireless Network Coding

- Sigcomm 2006: http://dx.doi.org/10.1145/1151659.1159942
  - COPE

- Objectives: practically applying network coding to unicast flow transmissions in a wireless network
  - Work with both UDP and TCP
- Key challenge: integration of network coding with the network stack
- Inter-flow coding
COPE: Key Design Decisions

- Use network coding: simple XOR operations
COPE: Key Design Decisions

- Opportunistic coding: inter-flow network coding
  - Gain knowledge of what neighbors have already heard
  - XORs (more than two) packets for different destinations and transmit a single packet if it knows that each destination has enough information to decode the encoded packet

- 3 main components
  - Opportunistic listening
  - Opportunistic coding
  - Learning neighbor state
COPE: Opportunistic Listening

- Wireless: broadcast medium
- Node use promiscuous mode
  - Store all packets for a limited period $T$ ($T=0.5$ s)

- Node broadcast reception report
  - Piggybacked on data packets
  - Or periodic control packets if no data
**COPE: Opportunistic Coding**

- What packets to code to maximize throughput?

<table>
<thead>
<tr>
<th>Queue at B</th>
<th>Next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>A</td>
</tr>
<tr>
<td>P2</td>
<td>C</td>
</tr>
<tr>
<td>P3</td>
<td>C</td>
</tr>
<tr>
<td>P4</td>
<td>D</td>
</tr>
</tbody>
</table>

Output queue: P4, P3, P2, P1

- Pool: P4, P1
- Pool: P3, P1
- Pool: P4, P3

- To transmit $n$ packets, $p_1, \ldots, p_n$, to $n$ nexthops, $r_1, \ldots, r_n$, a node can XOR the $n$ packets together only if each next-hop $r_i$ has all $n-1$ packets $p_j$ for $j \neq i$. 
COPE: Learning Neighbor States

- How does a node know which packet a neighbor already received?
  - How to avoid suboptimal coding decisions?
  - How to learn that A has P4 and P3, C has P4 and P1, D has P3 and P1?

- Reception reports are not reliable
  - Lost because of congestion or late because of low traffic

- Heuristic: use delivery probabilities of the link between the packet’s previous hop and the neighbor
  - Obtained through usual ETX measurements
COPE: Implementation

- Implemented between IP/routing and MAC
- **Components**
  - Packet coding algorithm
  - Packet decoding
  - Pseudo-broadcast
  - Hop-by-hop ACK and retransmission
  - Preventing TCP reordering
COPE Implementation: Header Format

Necessary for decoding at neighbors

Variable length header

Packets XOR-ed together

REPORT_NUM
SRC_IP | LAST_PKT | Bit Map

ACK_NUM

ACK Block

LOCAL_PKT_SEQ_NUM

NEIGHBOR | LAST_ACK | Ack Map

MAC Header
COPE Header
Routing Header (Optional; depends on protocol)
IP Header
COPE Implementation: Packet Encoding

- Design choice 1: never delay a packet
  - If the MAC is ready, send the packet immediately

- Design choice 2: XOR packets of similar size
  - Two sizes: small and large
  - To remove padding: check IP header

- Design choice 3: per-neighbor queuing
  - Two queues per-neighbor
  - Virtual queues (pointers)
  - Limit reordering: only head of virtual queues

- If node wants to encode $n^{th}$ packets
  - Make sure that $P_D = P_1 \times P_2 \times \ldots \times P_n > G$ (for instance, $G=0.8$)
  - Fairness: iterates over each neighbor with random permutation
COPE Implementation: Packet Decoding

- Packet pool with copy of overheard packets
  - Stored in hash table with key = packet ID
  - Garbage collected regularly

- If packet received
  - Retrieved all XORed packet and XOR them to decode
COPE Implementation: Pseudo-Broadcast

- IEEE 802.11 MAC
  - Unicast with reliable ACK mechanism and back-off mechanism
  - Broadcast: no ACK mechanism, no back-off mechanism

- Pseudo-broadcast
  - Abuse 802.11 unicast to benefit from reliability and backoff
  - Choose one of the neighbor as HW destination address
  - Other receivers are in promiscuous mode and can use the COPE header
COPE Implementation: Hop-by-Hop Ack and Retransmission

- **Hop-by-hop ack**
  - Complement IEEE 802.11 unicast ack: necessary for all other neighbors than destination HW address
  - Ack piggybacked: sent in COPE header
  - Asynchronous
  - If no data: dedicated control packet

- **Retransmission mechanism**
  - Retransmission timer $T_a$
  - If expires: packet put at heads of queue (can be coded)
COPE Implementation: Preventing TCP Reordering

- Asynchronous ACK can cause reordering
  - TCP can believe congestion takes place

- TCP ordering agent on each host
  - Ignores all packet not destined for particular host
  - For each TCP flow ending at host: maintains a packet buffer and reorder the last seq. numbers if necessary
    - Timer
COPE Control Flow

1. **Can send**
   - Dequeue head of Output Queue
   - **Encode if possible**
     - **Encoded?**
       - yes: **Schedule retransmissions**
       - no: **Add reception reports**
         - Add acks to header
         - To wireless device
     - **Encoded?**
       - yes: Add to Packet Pool
         - Decoded?
           - yes: Decode and schedule acks
             - yes: Decodable?
               - yes: Extract Reception Reports
                 - Update Neighbor’s State
                   - Update retransmission events
                   - Extract acks meant for me
                     - Packet arrival
           - no: Add to Packet Pool
         - no: Add to Packet Pool
       - no: Am I destination?
         - yes: Deliver to host
         - no: Am I nexthop?
           - yes: Add to Packet Pool
COPE in Ad-Hoc Network: TCP
COPE in Ad-Hoc Network: UDP

- No hidden nodes

![Graph showing network throughput vs. offered load with and without COPE]
COPE in a Mesh Network

![Graph showing Throughput Gain vs Ratio of uplink to downlink traffic]
COPE Discussion
Trading Structure For Randomness in Wireless Opportunistic Routing

- Sigcomm 2007: [http://dx.doi.org/10.1145/1282380.1282400](http://dx.doi.org/10.1145/1282380.1282400)

- **Objective:** apply random network coding to opportunistic routing scenario
  - Replace ExOR structured approach by a random one using network coding
  - Goal is to increase spatial reuse
  - MORE

- **Key challenges:** low-complexity, remain compatible with IEEE 802.1
  - How many packet to send: to make sure to reach the destination
  - When to stop sending: to make sure the destination receives enough packets to decode
  - How to code efficiently: to keep complexity low

- **Intra-flow coding**
MORE: Key Design Decisions

- Intra-flow network coding

- File-based (batch operation) and per-flow/batch state
MORE: Mode of Operation

- **Source:** operates on batches
  - File broken in batches of K packets (innovative packets)
  - Random linear network coding (code vector)
  - Forwarder list: ETX computation, nodes closer to the destination
  - Per batch: send coded packets until ACK from destination is received
MORE: Mode of Operation

- **Forwarder**
  - Checks whether on the forwarder list
  - Checks whether the packet is “innovative”: linear independence assumption (Gaussian elimination)
  - Per batch: stores innovative packet, drops others and re-encode

- **Destination**
  - Checks “innovation” of packet
  - When more than K packets received: start decoding
  - Send ACK using best-path routing (MORE built on top of 802.11)
MORE: How Many Packets Should a Forwarder Send?

- How to ensure that at least one node closer to the destination receives a packet
- Heuristic: can be computed in a distributed fashion

\[ z_j = \frac{L_j}{(1 - \prod_{k<j} \epsilon_{jk})} \]

- \( L_j \): expected number of packets that node \( j \) must forward
- \( \epsilon_{jk} \): loss probability between \( i \) and \( k \)
- \( z_j \): expected number of transmissions of node \( j \)

- Implementation within 802.11:
  - \( TX\_credit \)
  - When node receive packet from upstream: increase credit counter by \( TX\_credit \); when send: decrease credit counter; when credit_counter negative: stop sending
  - \( z_j \) used for pruning
MORE: When to Stop Transmission

- As soon as destination receives $K^{th}$ innovative packet: send ACK
  - Priority over data
  - Sent on the shortest path
  - Forwarder that overhear ACK stop transmission

- Forwarder are controlled by the credit counter
  - And timeout
  - Arrival of new batch flushes previous batch
MORE: “Fast” Network Coding

- Code only innovative packets
- Work on code vectors
  - Is sufficient to check linear independence
- Pre-code packets
  - When medium is unavailable
  - Encoding can be done iteratively for each new innovative packet
MORE: Header Format

- Header above the MAC layer

![Diagram showing header format with fields like PACKET_TYPE, SRC_IP, DST_IP, FLOW_ID, BATCH_NO, CODE VECTOR, NUM_FORWARDS, FORWRDER_ID, FRD_CREDIT, and encoded data.]
MORE: Control Flow

- Can transmit
  - Select backlogged flow (credit counter > 0)
    - Add header to the pre-encoded packet
      - Am I source?
        - yes
          - credit counter = 1
            - Send to device
              - Pre-encode new packet
        - no
          - credit counter = 1
          - Send to device
          - Pre-encode new packet
      - no
        - Discard packet

- Received packet
  - Lookup flow state
    - Am I pruned?
      - yes
      - Discard packet
      - no
        - Is my batch no. same?
          - older
            - no
              - Is my batch current?
                - yes
                  - PrehopETX > MyETX?
                    - yes
                      - credit counter += TX.cred
                        - Is packet innovative?
                          - yes
                            - Store packet
                            - Am I destination?
                              - yes
                                - Update pre-encoded packet
                                - Stored K packets?
                                  - yes
                                    - Queue Ack
                                    - Decode batch and deliver
                                  - no
                                    - Discard packet
                          - no
                            - Discard packet
          - no
            - Discard packet
          - yes
            - Discard packet
MORE: Discussion

- Fair medium access
Analog Network Coding

- Alice and Bob send simultaneously
- Router receives the sum of the two signals (plus time and phase shifts)

2 Time Slots $\rightarrow$ Even Higher Throughput
Challenges

- Interfered signal is not really the sum
  - Channel distort signal
  - Two signals are never synchronized
  - It is not $A(t) + B(t)$ but $f_1(A(t)) + f_2(B(t-T))$
Solution: Exploit Asynchrony

- Alice uses non-interfering bits from her signal to estimate her channel
- Alice compensates for her interfering signal

We exploit the lack of synchronization!
Opportunistic Transmission for Infrastructure Networks

- SOFT: http://dx.doi.org/10.1145/1287853.1287871
- SoftRepeater: http://dx.doi.org/10.1109/TNET.2009.2026414
- FatVAP: http://www.usenix.org/event/nsdi08/tech/kandula.html
What is an Infrastructure Network?

- Typical examples
  - IEEE 802.11 enterprise network/hotspot deployment: multiple APs connected by backbone (Ethernet)
  - Cellular network: multiple BSs connected by backbone

![Diagram of Infrastructure Network]

- Traditional: client associated with single AP
Opportunistic Reception and Combining

- Opportunistic reception: packet can be received by neighboring AP
- Combining (SOFT): multiple erroneous packets can be combined to obtain correct output
  - How?
Opportunistic Forwarding

- A neighbor might help in case of packet loss (SoftRepeater)
- Or two hops may provide better performance than one
- Can even use network coding!
Discussion

- **Backbone**
  - Throughput (if soft values) and delay requirements
  - Synchronization

- **TCP friendliness**
  - Reordering, jitter

- **IEEE 802.11: How much does the MAC need to be redesigned?**
  - Rate adaptation
  - Acknowledgement mechanism
  - Retransmission mechanism
Building Blocks of Wireless Networking

- Rate control
- Power control
- Per-queue
- Backpressure operation
- Opportunistic transmissions
- Piggyback ACK
- Network coding
- ARQ protocols
TCP over Wireless Networks: Problems

- Wireless links are inherently error-prone
  - Fades, interference, attenuation
  - Errors often happen in bursts
- TCP cannot distinguish between corruption and congestion
  - TCP unnecessarily reduces window, resulting in low throughput and high latency
- Burst losses often result in timeouts
- Sender retransmission is the only option
  - Inefficient use of bandwidth
- Interference between two-way traffic (TCP ack)
TCP over Wireless Networks: Problems

- Disambiguating wireless bit-errors from congestion
  - Frequent window reduction due to errors
  - Frequent timeout due to burst losses

- High variability
  - Contention + channel errors: high loss
  - Contention + link-layer retransmission: high RTT variance

- Need end-to-end connection all the time
  - Wireless links have intermittent connectivity
TCP over Wireless Networks: Performance Degradation

2 MB wide-area TCP transfer over 2 Mbps Lucent WaveLAN

Best possible TCP with no errors (1.30 Mbps)

TCP Reno (280 Kbps)
TCP over Wireless Links: Possible Solutions

- **Incremental deployment**
  - Solution should not require modifications to fixed hosts
  - If possible, avoid modifying mobile hosts

- **End-to-end protocols**
  - Selective ACKs, explicit loss notification

- **Split-connection protocols**
  - Separate connections for wired path and wireless hop

- **Reliable link-layer protocols**
  - Error-correcting codes
  - Local retransmission

- **TCP aware link aware**

- **Network assisted congestion control**
End-to-End Modifications

- Improve TCP implementations
  - Not incrementally deployable
  - Improve loss recovery (SACK, NewReno)
  - Help it identify congestion (ELN, ECN)
    - ACKs include flag indicating wireless loss
  - Trick TCP into doing right thing: E.g. send extra dupacks
Link Layer Approaches

- More aggressive local retransmit than TCP
  - Bandwidth not wasted on wired links
- Possible adverse interactions with transport layer
  - Interactions with TCP retransmission
  - Large end-to-end round-trip time variation
- FEC does not work well with burst losses
Link Layer: ARQ

Internet

FH

BS
MH

Packet 1
Packet 2
Packet 3

Link-Ack 1
Link-Ack 2
Link-Ack 3

Timeout

Packet 3
Link Layer: ARQ, Discussion

- **Pros**
  - No modification to upper layers

- **Cons**
  - Fast retransmission due to message lost and out-of-order delivery
    - redundant retransmission and window reduction
  - Interacts with TCP retransmissions
    - redundant retransmission
  - Large RTT variance
    - long timeout
  - Head-of-line blocking due to large #retransmission
    - slow link blocks fast link
Split-TCP: Indirect TCP (I-TCP)

- I-TCP splits end-to-end TCP connection into two connections
  - Fixed host to BS
  - BS to mobile host
- Two TCP connections with independent flow/congestion control contexts
- Packets buffered at BS
Split-TCP: Discussion

- **Pros**
  - Separates flow and congestion control of wireless and wired
  - Higher throughput at sender

- **Cons**
  - Breaks TCP end-to-end semantics
    - Ack at FH does not mean MH has received the packet
  - BS failure causes loss of data
    - Neither FH nor MH can recover the data
  - On path change, data has to be forwarded to new BS
  - Wireless part is the bottleneck
**Link Layer Aware: SNOOP**

- Link layer is aware of TCP traffic
- BS caches data and monitors acks. Retransmits on duplicate acks and drops duplicate acks

SNOOP: Discussion

- **Pros**
  - No modification to FH and MH
  - BS only keeps soft state—BS failure does not break TCP

- **Cons**
  - Does not work with encrypted packets
  - Does not work if data packets and acks traverse different paths
  - Increases RTT—high timeout
Discussion

- Many assumptions built into Internet design
  - Wireless forces reconsideration of issues

- Most above protocols focus on “the last mile” problem—single wireless hop
  - Multi-hop wireless mesh networks are blooming, e.g., Roofnet, Wildnet, …

- None of above protocols works during network partitions

- Transport
  - Losses can occur due to corruption as well as congestion
    - Impact on TCP?
  - How to fix this? Hide it from TCP or change TCP