Advanced Distributed Algorithms and Data Structures

Chapter 6: Contention Resolution

Christian Scheideler
Institut für Informatik
Universität Paderborn

Overview

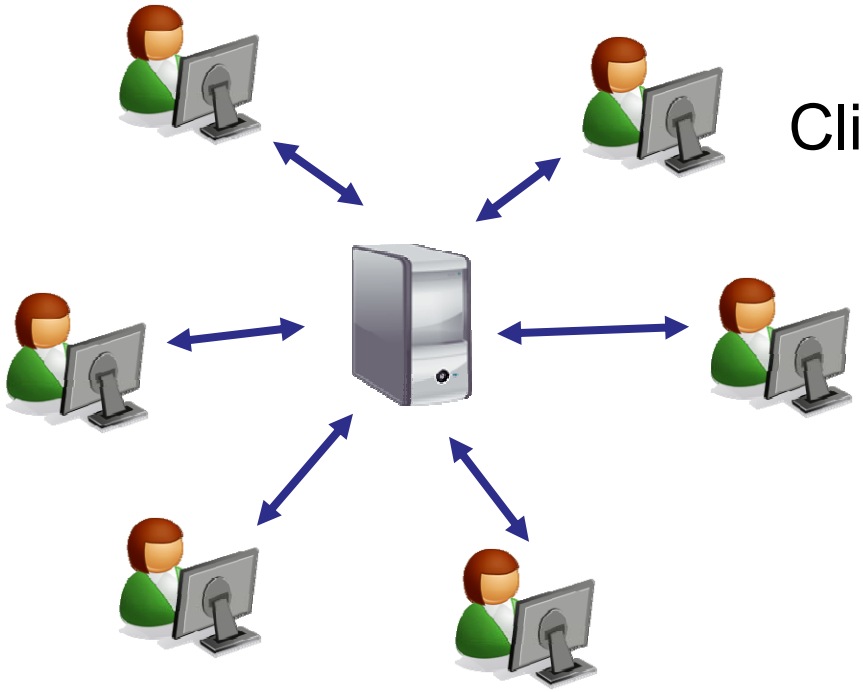
- Motivation and problem
- Contention resolution in the Internet
- Contention resolution in wireless networks

Motivation

- In large distributed systems, failures are not an exception but the norm.
- To find out about failed/killed processes, each process should periodically check its neighbors.
- How often should a process check its neighbors to avoid contention problems?

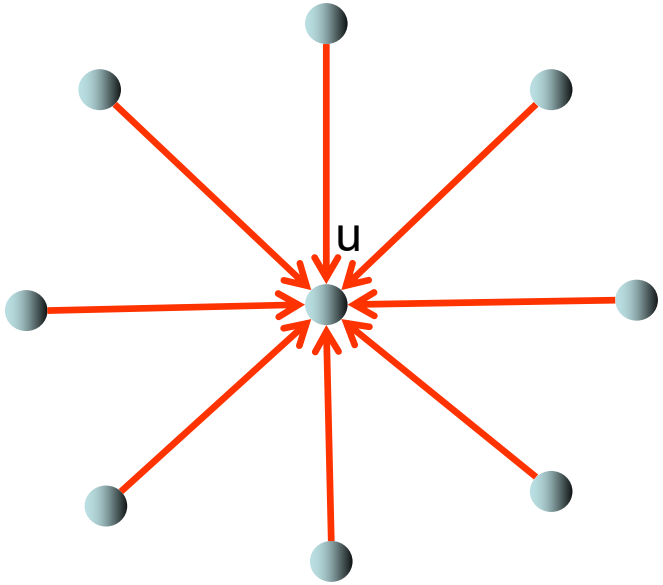
Contention Resolution Problem

Concrete scenario: n clients want to periodically ping some server



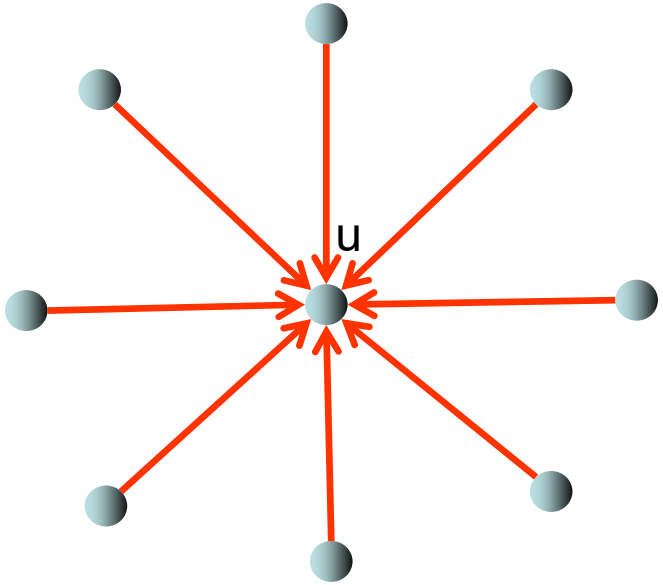
Contention Resolution Problem

Abstract setting: n nodes want to periodically ping some node u



Contention Resolution Problem

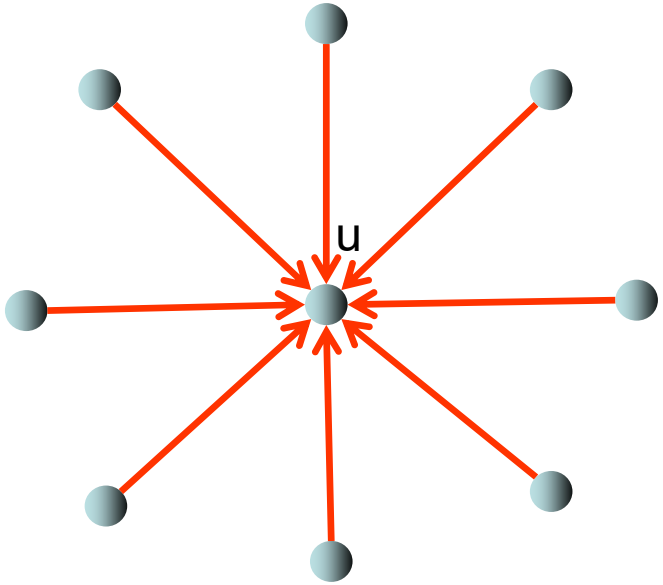
Problem: If all of the nodes sind a ping message to u in each time step, u gets overwhelmed with messages.



Contention Resolution Problem

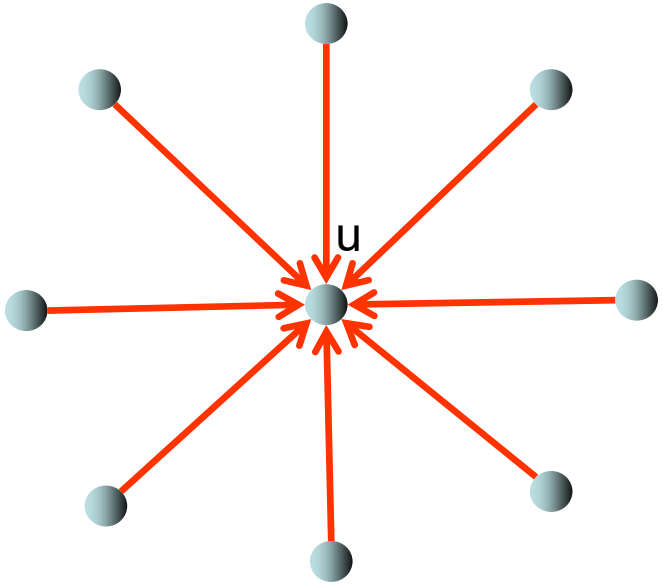
Ideal: on average, u only gets a constant number of ping messages in each round.

$\rightarrow$ each node should ping u with probability 1/n



Contention Resolution Problem

To converge to a ping probability of 1/n, we need a contention resolution protocol.



Overview

- Motivation and problem
- Contention resolution in the Internet
- Contention resolution in wireless networks

Contention Resolution

Popular approach in the Ethernet: exponential backoff (Metcalfe and Boggs)

Instead of a node u, we have a shared medium, and only one computer can use the medium at a time.

Contention Resolution

Assumptions:

- A node can only successfully tansmit a message on the shared medium if it is the only one currently transmitting a message.
- If more than one node attempts to transmit a message at the same time, a collision occurs.
- A node attempting to transmit a message can detect whether a collision occurs or not. (This allows so-called carrier sense multiple access with collision detection (short CSMA/CD) protocols to be used.)
- Time is divided into synchronized time slots, and it takes just one time slot to transmit a message.

Exponential backoff protocol:

- after c collisions, a random time slot between 0 and 2^c-1 is chosen for the next transmission attempt

Contention Resolution

Exponential backoff protocol:

- after c collisions, a random time slot between 0 and 2^c-1 is chosen for the next transmission attempt

Theorem 6.1: Given n nodes where node v had $c(v)$ collisions in the past, and every node only wants to transmit one message, the expected time of node v to succeed in transmitting its message is $O(\max\{n,2^{c(v)}\})$.

Proof:

- Let p_v denote the transmission probability of node v. Node v needs at most $\sum_{i=1}^{\log n} 2^i \leq 2n$ many time steps till $p_v \leq 1/n$.
- In general, after at most $\sum_{i=1}^{(\log n)+j} 2^i \leq 2n \cdot 2^j$ many time steps, $p_v \leq 1/(n \cdot 2^j)$.
- Hence, after $2n$ steps, the probability of the j-th transmission attempt of node v to fail is at most $(n-1) \cdot 1/(n \cdot 2^{j-1}) \leq 1/2^{j-1}$.
- Thus, the probability that, after $2n$ steps, v needs at least j transmission attempts, is at most

$$\prod_{i=1}^{j-1} 1/2^{i-1} \leq 1/2^{(j-2)^2/2}$$

- Exercise: bound based on that the expected number of transmission attempts of a node and from that the expected time for a node to succeed.

Contention Resolution

Problem with exponential backoff:

- No convergence to 1/n. Instead, for each new message a new backoff process is started. This may create bursts of messages at certain times, which is problematic for other environments like the Internet.

Alternatives:

- additive increase multiplicative decrease (AIMD) mechanism to adjust the probabilities, which is similar to TCP
- Multiplicate increase multiplicative decrease (MIMD) mechanism to adjust the probabilities

AIMD Protocol 1

AIMD protocol 1:

- Initially, set T to some fixed value T_{min} (determined by the maximum number of new nodes to be expected in a time slot).
- Decide with probability 1/T to ping node u.
- If so, consider the following two cases:
  If u sends an ACK back, then set $T:=\max\{T-1,T_{min}\}$.
  Otherwise, set T:=2T.

Simplifying assumption:

For any sent ping message the ACK will be received instantly whenever at most c nodes ping u at the same time. (Otherwise, no ACK is sent by u.)

AIMD Protocol 1

- $T_v(t)$: T value of v at time t
- $p_v(t)=1/T_v(t)$: ping probability of v at time t

How does $p_v(t)$ change over the time?

$$E[p_v(t+1)] = E[1/T_v(t+1)]$$
$$= Pr[\text{ping}] \cdot (Pr[\text{ACK}] \cdot 1/(T_v(t)-1) + Pr[\text{no ACK}] \cdot (1/(2T_v(t))) + Pr[\text{no ping}] \cdot (1/T_v(t))$$
$$= (1/T_v(t)) \cdot (Pr[\text{ACK}] \cdot 1/(T_v(t)-1) + Pr[\text{no ACK}] \cdot (1/(2T_v(t))) + (1-1/T_v(t)) \cdot (1/T_v(t))$$

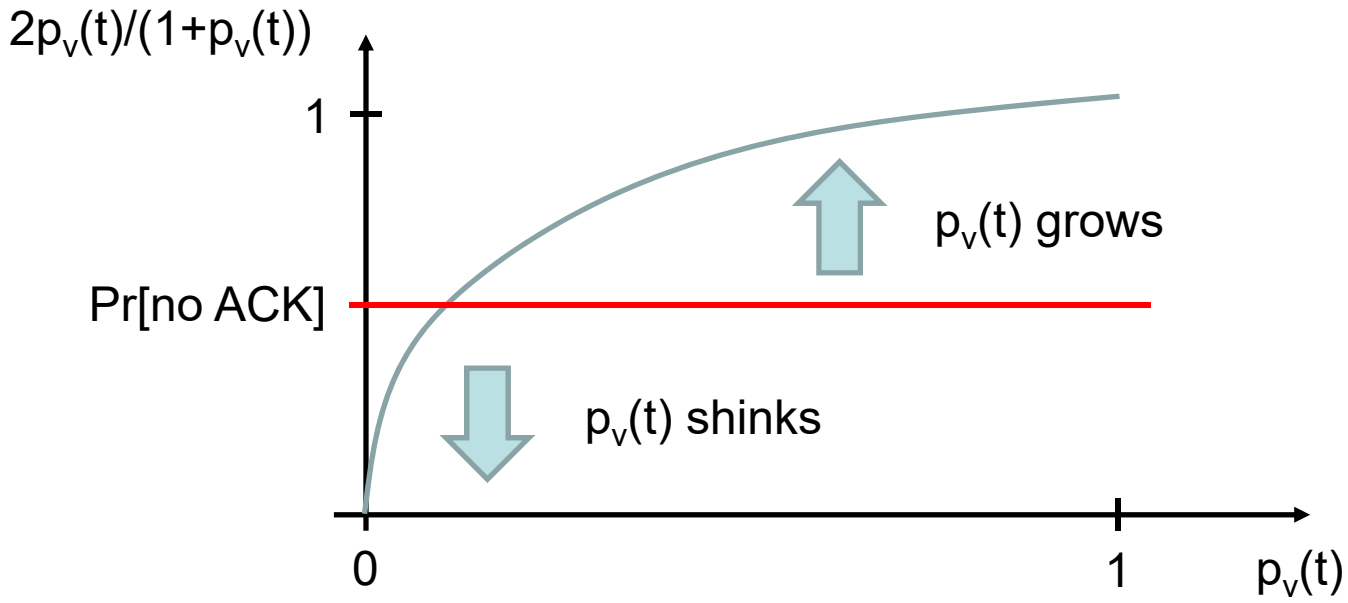
Hence, $E[p_v(t+1)] \geq p_v(t)$ if and only if

$$Pr[\text{ACK}] \cdot 1/(T_v(t)-1) + Pr[\text{no ACK}] \cdot (1/(2T_v(t)) \geq 1/T_v(t) \quad (*)$$

Let $q=Pr[\text{no ACK}]$. Then (*) holds if and only if $q \leq 2/(T_v(t)+1)$. Or in other words, $q \leq 2p_v(t)/(1+p_v(t))$.

This, however, is bad, as we will see.

AIMD Protocol 1



Hence, nodes with larger ping probabilities have a better chance of growing their probabilities than those with lower ping probabilities, which destroys fairness.

AIMD Protocol 2

AIMD protocol 2: (similar to TCP)

- Initially, set k to 1. N is assumed to be a sufficiently large value known to all nodes so that it upper bounds n.
- Decide with probability k/N to ping node u.
- If so, consider the following two cases:
  If u sends an ACK back, then set k:=min{k+1,N}.
  Otherwise, set k:=max{k/2,1}.

Simplifying assumption:

For any sent ping message the ACK will be received instantly whenever at most c nodes ping u at the same time. (Otherwise, no ACK is sent by u.)

AIMD Protocol 2

- $k_v(t)$: k value of v at time t
- $p_v(t)=k_v(t)/N$: ping probability of v at time t

How does $p_v(t)$ change over the time?

$$
\begin{aligned}
E[p_v(t+1)] &= E[k_v(t+1)/N] \\
&= Pr[\text{ping}] \cdot (Pr[\text{ACK}]\cdot(k_v(t)+1)/N + Pr[\text{no ACK}]\cdot k_v(t)/(2N)) + \\
&\quad Pr[\text{no ping}] \cdot (k_v(t)/N) \\
&= (k_v(t)/N) \cdot (Pr[\text{ACK}]\cdot(k_v(t)+1)/N + Pr[\text{no ACK}]\cdot k_v(t)/(2N)) + \\
&\quad (1-k_v(t)/N) \cdot (k_v(t)/N)
\end{aligned}
$$

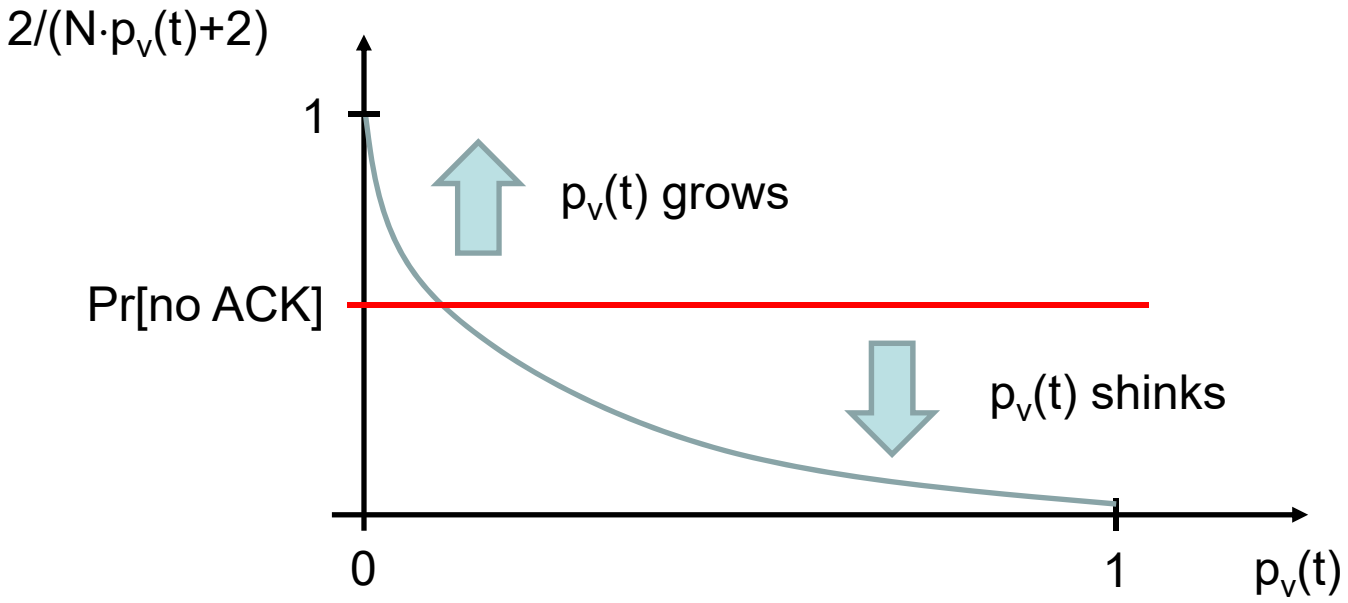
Hence, $E[p_v(t+1)] \geq p_v(t)$ if and only if

$$Pr[\text{ACK}]\cdot(k_v(t)+1)/N + Pr[\text{no ACK}]\cdot k_v(t)/(2N) \geq k_v(t)/N \qquad (*)$$

Let $q=Pr[\text{no ACK}]$. Then (*) holds if and only if $q \leq 2/(k_v(t)+2)$. Or in other words, $q \leq 2/(N\cdot p_v(t)+2)$.

This is better, as we will see.

AIMD Protocol 2



Now, nodes with smaller ping probabilities have a better chance of growing, so fairness can be achieved. However, for the probabilities to converge to a sum of 1 (which would be ideal), Pr[no ACK] would have to be equal to 2n/N. This would require u to set its threshold c to ~log(N/n). Also, the convergence to 1/n for each node is slow due to the additive increase. Can we do better?

MIMD Protocol

MIMD protocol:

- Initially, set p to p_{max} for some constant $p_{max}<1$ (which depends on the maximum number of new nodes to be expected in a time step). Let $0<\gamma\leq 1$ be a sufficiently small constant.
- Decide with probability p to ping node u.
- If so, consider the following two cases:
  If u sends an ACK back, then set $p:=\min\{(1+\gamma)p,p_{max}\}$.
  Otherwise, set $p:=p/(1+\gamma)$.

Simplifying assumption:

For any sent ping message the ACK will be received instantly whenever at most c nodes ping u at the same time.

MIMD Protocol

- $p_v(t)$: p value of v at time t

How does $p_v(t)$ change over the time?

$$E[p_v(t+1)] = \Pr[\text{ping}] \cdot (\Pr[\text{ACK}]\cdot(1+\gamma)p_v(t) + \Pr[\text{no ACK}]\cdot p_v(t)/(1+\gamma)) + \Pr[\text{no ping}] \cdot p_v(t)$$
$$= p_v(t) \cdot (\Pr[\text{ACK}]\cdot(1+\gamma)p_v(t) + \Pr[\text{no ACK}]\cdot p_v(t)/(1+\gamma)) + (1-p_v(t)) \cdot p_v(t)$$

Hence, $E[p_v(t+1)] \geq p_v(t)$ if and only if

$$\Pr[\text{ACK}]\cdot(1+\gamma)p_v(t) + \Pr[\text{no ACK}]\cdot p_v(t)/(1+\gamma) \geq p_v(t) \quad (*)$$

Let $q=\Pr[\text{no ACK}]$. Then (*) holds if and only if $q \leq (1+\gamma)/(2+\gamma)$.

For $\Sigma_{v\in V} p_v$ to converge to 1, the threshold c in u should be set so that $\Pr[\text{no ACK}] = (1+\gamma)/(2+\gamma)$ whenever $\Sigma_{v\in V} p_v=1$. A good approximation for that is $c=1$ since then it holds for any $v\in V$ that

$$\Pr[\text{no ACK}] = 1- \Pi_{w\neq v}(1-p_w) \approx 1-e^{\Sigma_{w\neq v} p_w} \approx 1-1/e \in [1/2,2/3]$$

MIMD Protocol

Problem: ACK needs more than one time step to be sent back

Solution: once the next ping transmission is attempted, check whether ACK of previous transmission has already arrived. If not, consider the previous transmission to have failed (i.e., no ACK was sent). Adapt the access probability accordingly.

MIMD Protocol

```
Subject MIMD_Client:
    server, in: Relay
    p: Real
    delay: Integer
    ack_received: Boolean

init(s) →
    server:=s
    in:=new Relay
    p:=p_max
    ack_received:=false
    delay:=random_time(p)
    enable(timeout, delay)
```

```
timeout: true →
    if ack_received then
        p:=min{(1+γ)p,p_max}
    else
        p:=p/(1+γ)
    server←ping(in)
    ack_received:=false
    delay:=random_time(p)
    enable(timeout, delay)

pong() →  { processes ACK}
    ack_received:=true
```

random_time(p) outputs $t \in \mathbb{N}$ with probability $(1-p)^{t-1} \cdot p$.

MIMD Protocol

```
Subject MIMD_Server:
    in: Relay
    counter: Integer
    Q: Queue of Relay

    init() →
        in:=new Relay
        counter:=0
        Q:=∅

    ping(out) →
        counter:=counter+1
        enqueue(Q,out)
```

```
timeout: true →
        if counter=1 then
            out:=dequeue(Q)
            out←pong() {sends ACK}
            delete out
        else
            while not empty(Q) do
                out:=dequeue(Q)
                delete out
        counter:=0
```

MIMD Protocol

Problem: the MIMD Protocol has no means to converge to fairness as every node v basically has the same Pr[no ACK] and therefore all p_v´s have the same drift direction. Even worse, if we start with a $\Sigma_v\, p_v << 1$, then the ratio $E[p_{t+1}(v)]/p_t(v)$ gets larger the larger the $p_t(v)$ values are. Hence, in this case larger probabilities are growing quicker than smaller ones.

Solution 1:

- Every ping of node v contains p_v (resp. the i with $p_v = p_{max}/(1+\gamma)^i$).
- Among the nodes contacting u between two timeouts, an ACK is sent back only for the node with smallest probability, and only if at most d nodes contacted u for some constant $d>1$.

MIMD Protocol

Problem: the MIMD Protocol has no means to converge to fairness as every node v basically has the same Pr[no ACK] and therefore all p_v´s have the same drift direction. Even worse, if we start with a $\Sigma_v\, p_v << 1$, then the ratio $E[p_{t+1}(v)]/p_t(v)$ gets larger the larger the $p_t(v)$ values are. Hence, in this case larger probabilities are growing quicker than smaller ones.

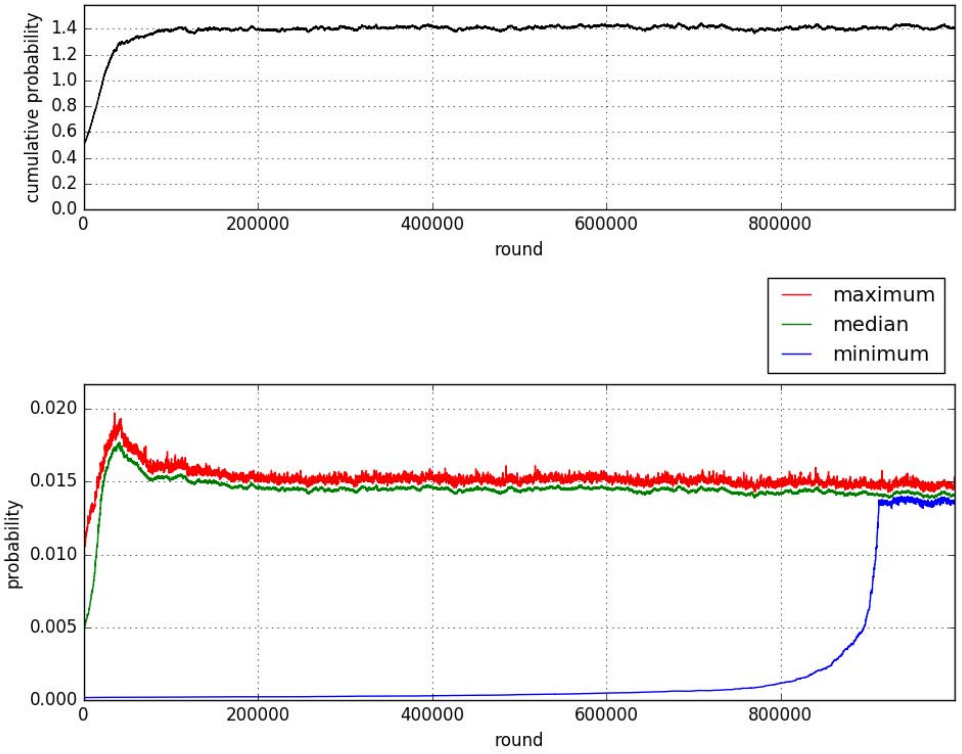
Solution 2 (Fair MIMD Protocol):

- Every ping of node v contains p_v (resp. the i with $p_v = p_{max}/(1+\gamma)^i$).
- Node u collects the past d successfully acknowledged ping messages for a sufficiently large, odd d and sends in its ACK the median value of the p_v´s of these pings back to the sending node, say v.
- Node v then resets its p_v to $p_v := (1+\gamma)\cdot \text{median}$.

Simulation of Strategy 1

Setup: 100 nodes, γ=0.01, d=3, just one run

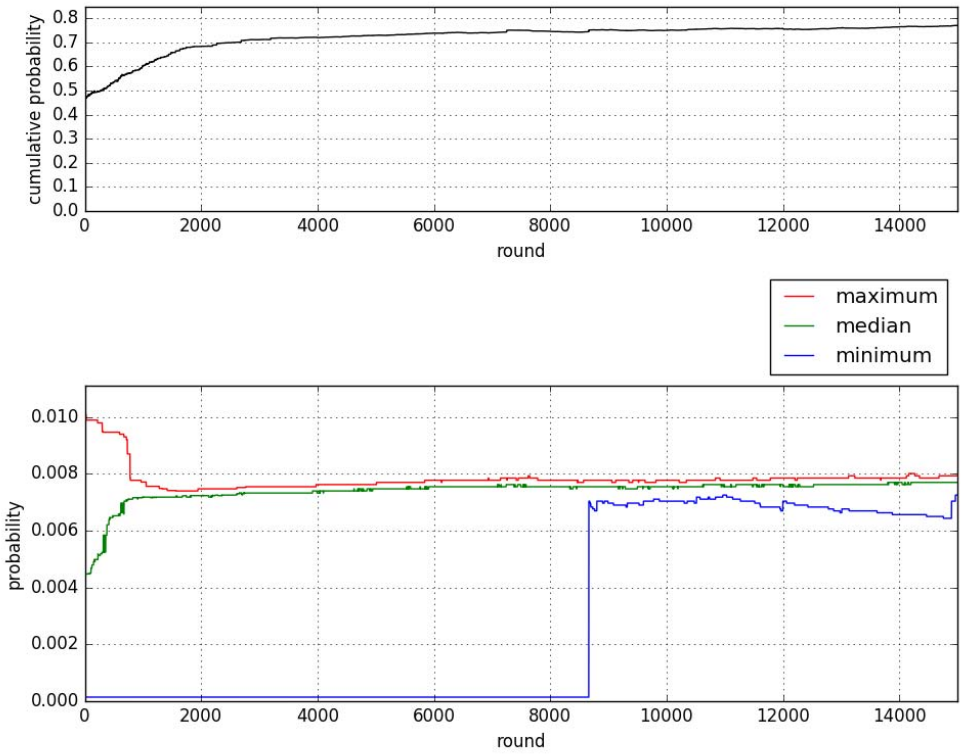
Strategy 1 is indeed fair, but it may take a long time to converge (if γ is small).



Simulation of Strategy 2

Setup: 100 nodes, γ=0.01, d=3, just one run

Strategy 2 is also fair and converges much faster than Strategy 1.



Fair MIMD Protocol

```
Subject Fair_MIMD_Client:
    server, in: Relay
    delay, i, med_i: Integer
    ack_received: Boolean

init(s) →
    server:=s
    in:=new Relay
    i:=0   { prob=p_max }
    ack_received:=false
    delay:=random_time(p(i))
    enable(timeout, delay)
```

We set $p(i)=p_{max}/(1+\gamma)^i$

```
timeout: true →
    if ack_received then
        i:=max{med_i-1,0}
    else
        i:=i+1
    server←ping(in, i)
    ack_received:=false
    delay:=random_time(p(i))
    enable(timeout, delay)

pong(median) →
    ack_received:=true
    med_i:=median
```

Fair MIMD Protocol

```
Subject Fair_MIMD_Server:
    in: Relay
    counter, i: Integer
    Q: Queue of Relay
    M: Queue of Integer

    init() →
        in:=new Relay
        counter:=0
        Q:=∅
        M:=∅

    ping(out, ival) →
        counter:=counter+1
        i:=ival
        enqueue(Q,out)
```

```
timeout: true →
        if counter=1 then
            out:=dequeue(Q)
            enqueue(M,i)
            if size(M)>3 then
                dequeue(M)
            out←pong(median(M))
            delete out
        else
            while not empty(Q) do
                out:=dequeue(Q)
                delete out
        counter:=0
```

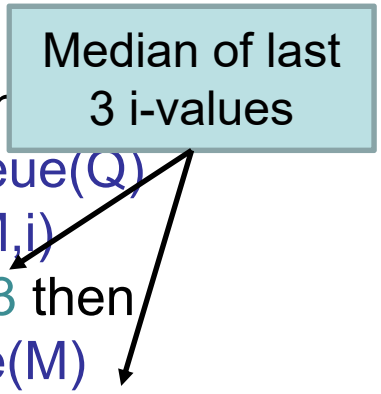
Fair MIMD Protocol

```
Subject Fair_MIMD_Server:
    in: Relay
    counter, i: Integer
    Q: Queue of Relay
    M: Queue of Integer

    init() →
        in:=new Relay
        counter:=0
        Q:=∅
        M:=∅

    ping(out, ival) →
        counter:=counter+1
        i:=ival
        enqueue(Q,out)
```

```
timeout: true →
        if counter=1 th
            out:=dequeue(Q)
            enqueue(M,i)
            if size(M)>3 then
                dequeue(M)
            out←pong(median(M))
            delete out
        else
            while not empty(Q) do
                out:=dequeue(Q)
                delete out
        counter:=0
```



Fair MIMD Protocol

Why median and not the average value?

The median is more robust to manipulations and extreme values (due to, for example, newly connected nodes).

No formal analysis of the Fair MIMD protocol is known yet.

Conjecture:

- Suppose that $p_1 \leq \ldots \leq p_n$ are the ping probabilities used by the nodes, sorted in increasing order, and let the median probability, p_{med}, be defined as the p_j where $\Sigma_{i<j}\ p_i \leq ½\ \Sigma_i\ p_i$ and $\Sigma_{i>j}\ p_i \leq ½\ \Sigma_i\ p_i$. Then the other probabilities converge towards p_{med}, ultimately resulting in fairness.

We will come back to the median rule later in a different context.

Fair MIMD Protocol

Application: mutual exclusion

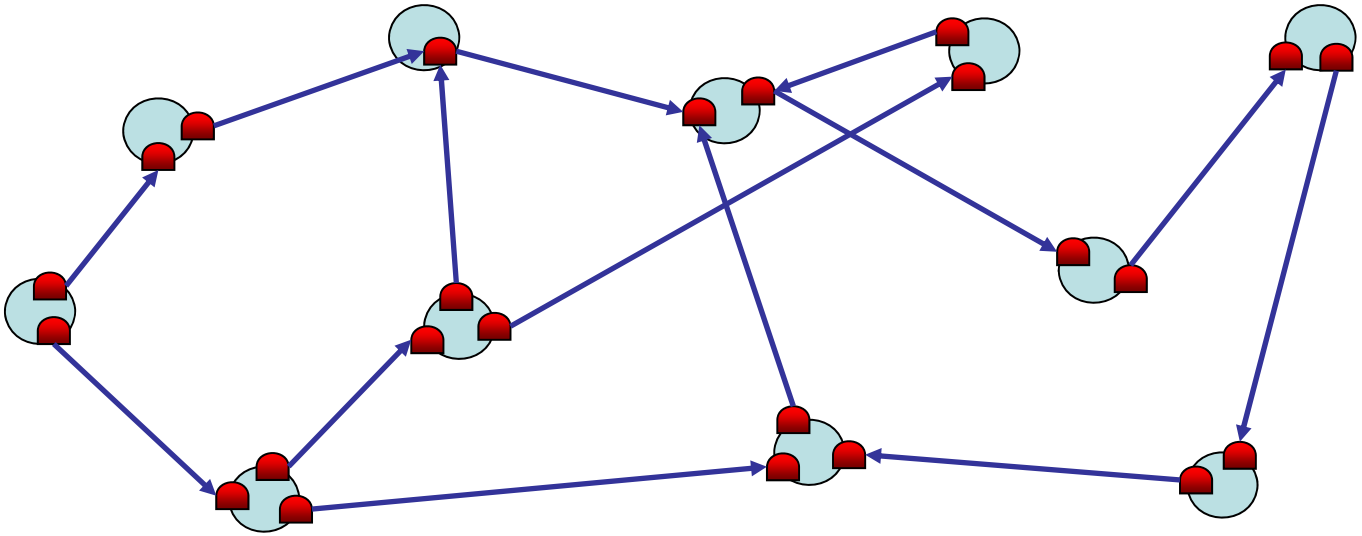
The mutual exclusion problem is a resource allocation problem in which n processes contend for exclusive access to a given resource. Any solution to this problem must satisfy the following two properties:

- Mutual Exclusion (safety):
  No two processes access the resource simultaneously.
- No Starvation (liveness):
  Provided that any process holding the resource will eventually release it, a process that requests the resource will eventually get it.

- The safety property can be handled by our contention resolution protocols: once exactly one process sends a ping to the resource, it will have the right to access it.
- The liveness property is satisfied if we have fairness, since in this case every process eventually has a probability of around 1/n to succeed.

General MIMD Protocol

Contention resolution problem for arbitrary relay graphs:

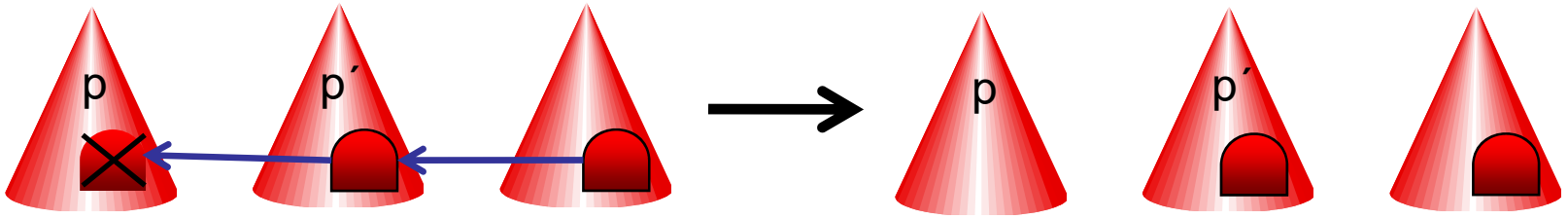


- Execute Fair_MIMD_Client Protocol in parallel for each outgoing relay (i.e., each such relay maintains prob p(i))
- Execute Fair_MIMD_Server Protocol in parallel for each process or sink relay

General MIMD Protocol

Application: monitoring of relays at the TCL layer

- If a relay r is deleted by a process, it is just marked for deletion in the TCL as long as it has incoming links (or outgoing messages). Once all incoming links are gone (and all outgoing messages are gone or cannot be sent further due to a missing outgoing link), the relay is deleted.
- A relay r is a dead end if it was deleted by a process or it has no incoming link any more (i.e., dead(r)=true).
- Whenever a ping is sent to a dead end r, a NACK message is sent back to indicate that r is a dead end.
- Whenever a TCL receives a NACK for some relay r, it marks it as a dead end.
- In this way, cascading deletions of incoming links can be realized.

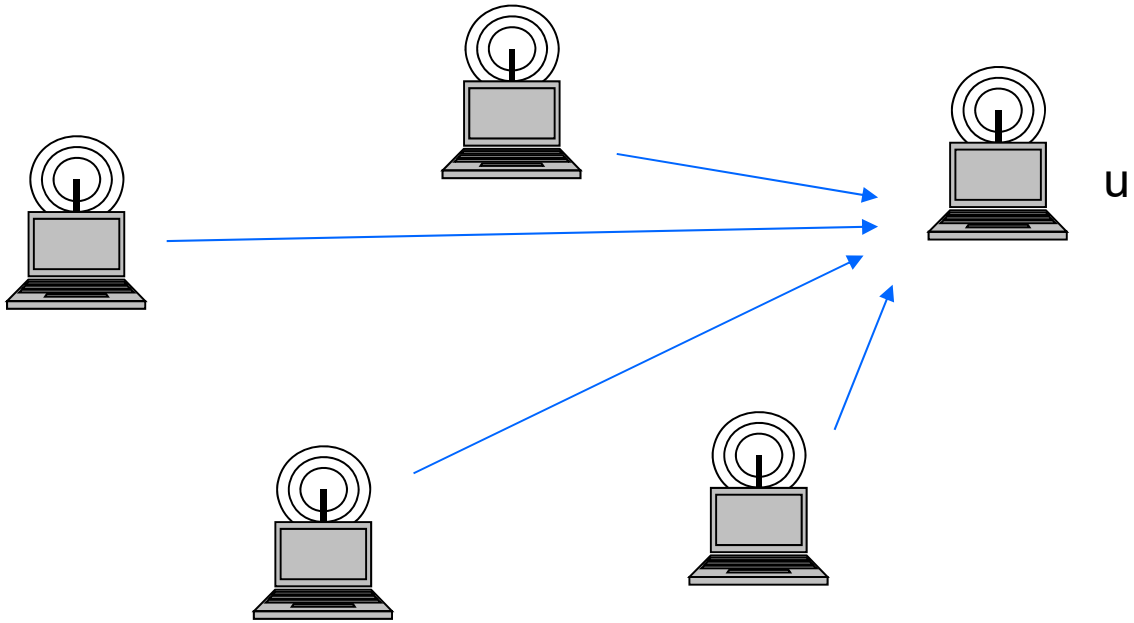


Overview

- Motivation and problem
- Contention resolution in the Internet
- Contention resolution in wireless networks

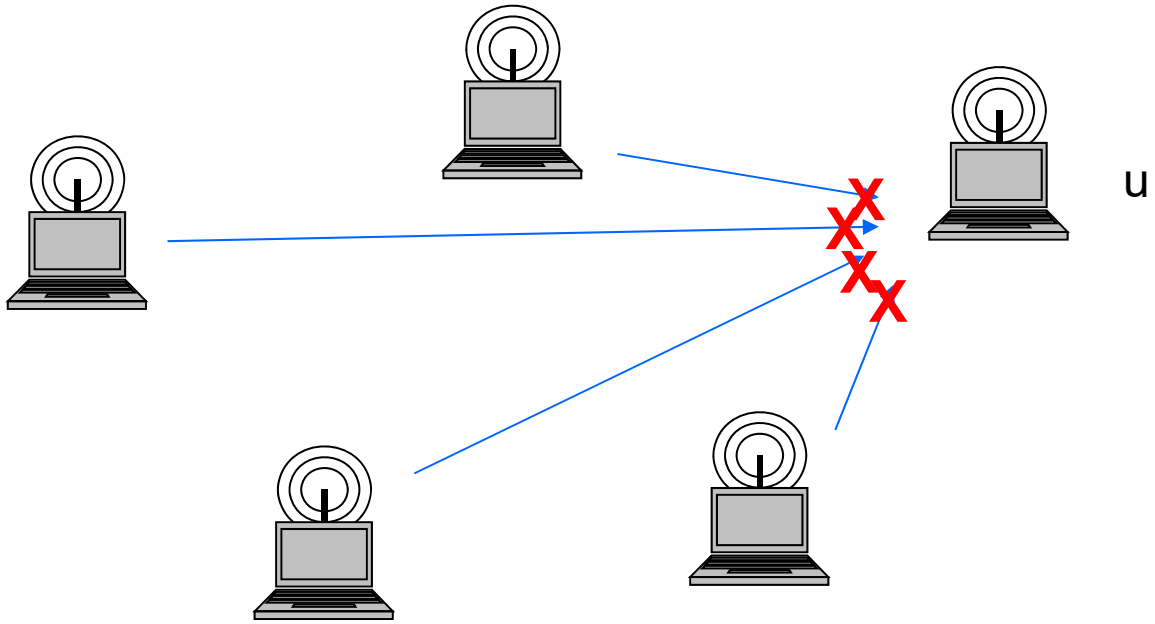
Wireless Contention Resolution

We assume that only one node can use the wireless medium at a time.



Wireless Contention Resolution

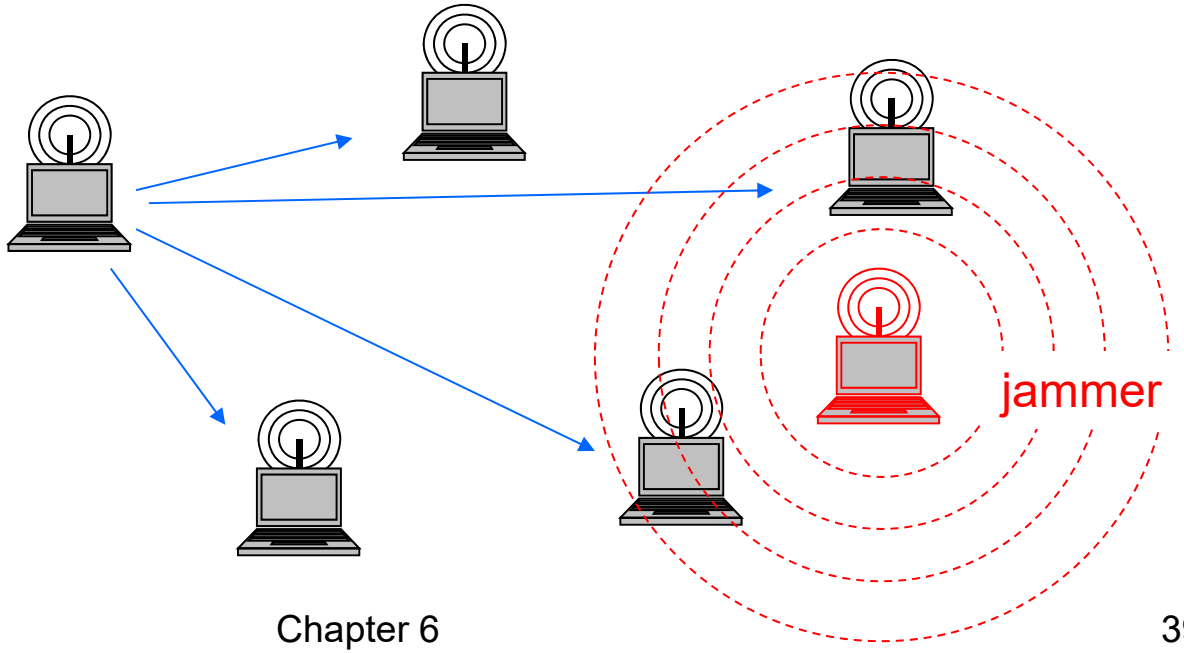
Many problems: wireless channel blocked due to interference, noise, or collisions



Adversarial Jamming

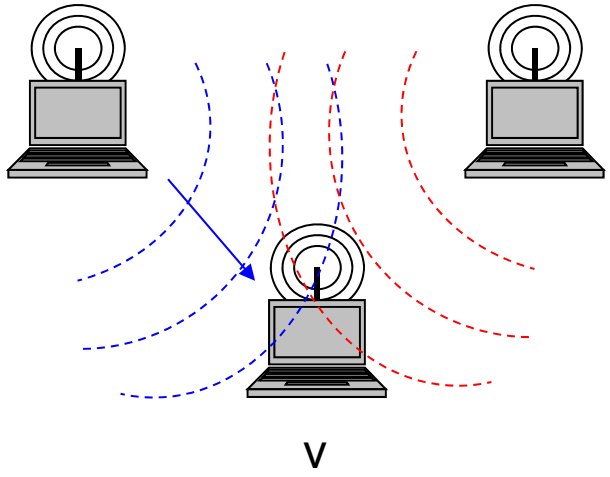
A jammer listens to the open medium and broadcasts in the same frequency band as the network

- no special hardware required
- can lead to significant disruption of communication at low cost for the jammer



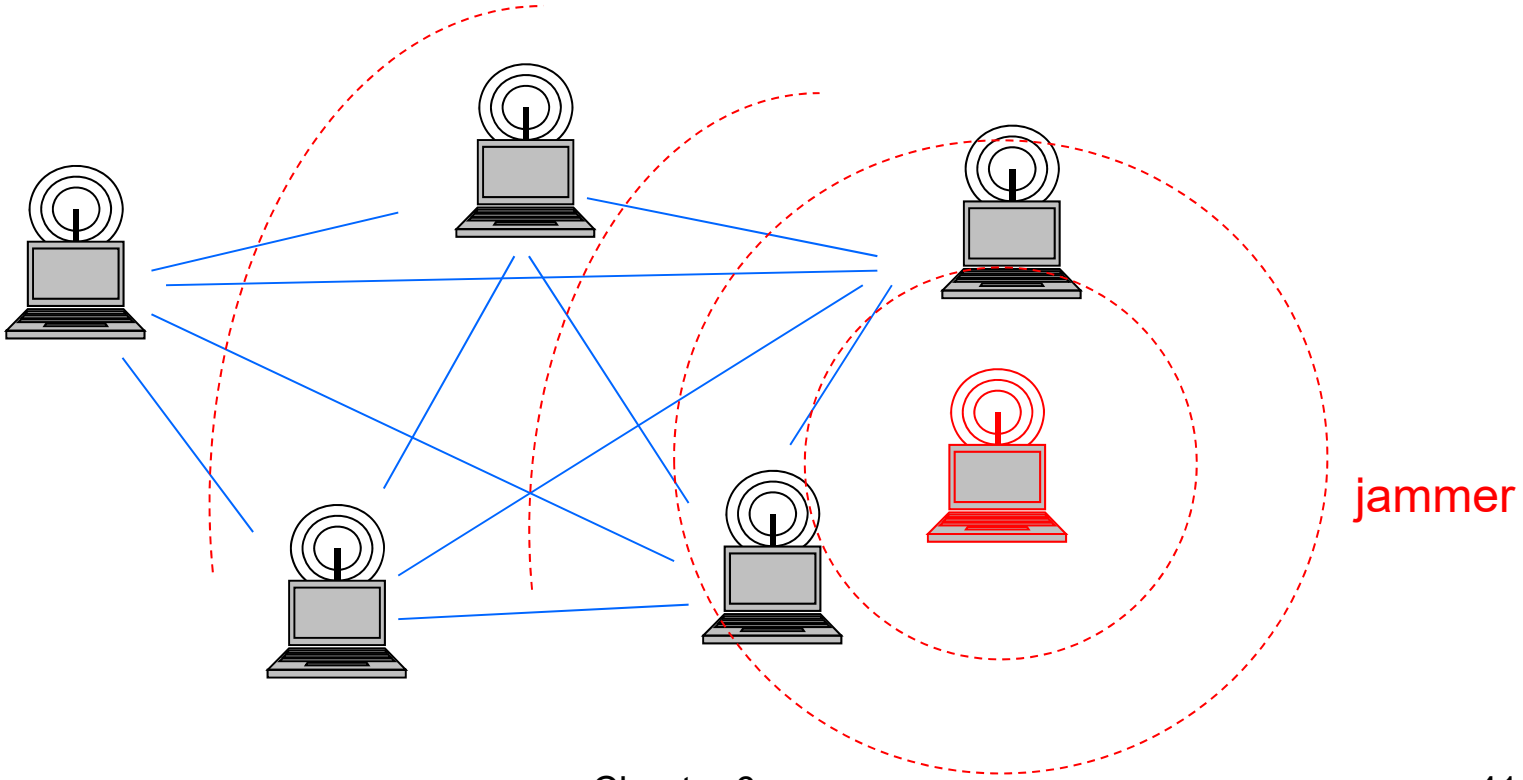
Wireless Communication Model

- at each time step, a node may decide to transmit a packet (nodes continuously contend to send packets)
- a node may transmit or sense the channel at any time step (half-duplex)
- when sensing the channel a node *v* may
  - sense an idle channel
  - receive a packet
  - sense a busy channel



Single-hop Wireless Network

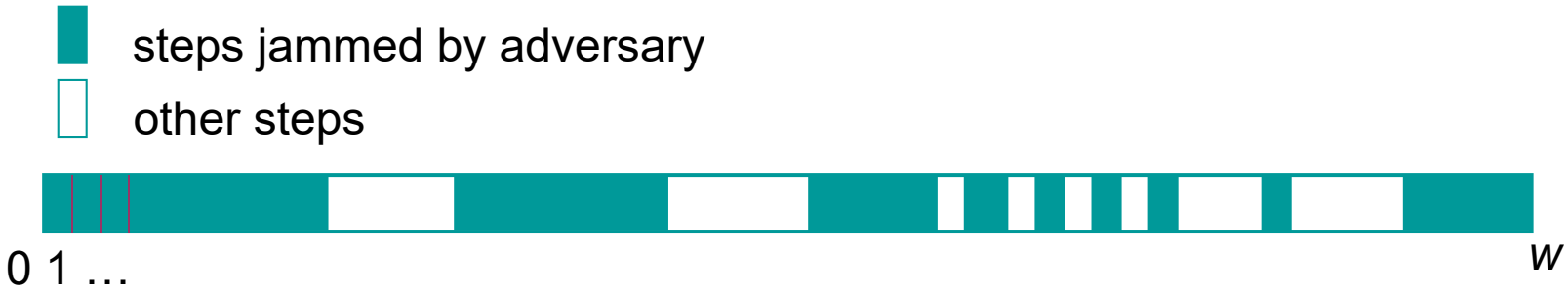
- *n* reliable honest nodes and one jammer; all nodes within transmission range of each other and of the jammer



Adaptive Adversary

- knows protocol and entire history
- nodes cannot distinguish between adversarial jamming or a message collision
  - i.e., a node senses a busy channel in both cases

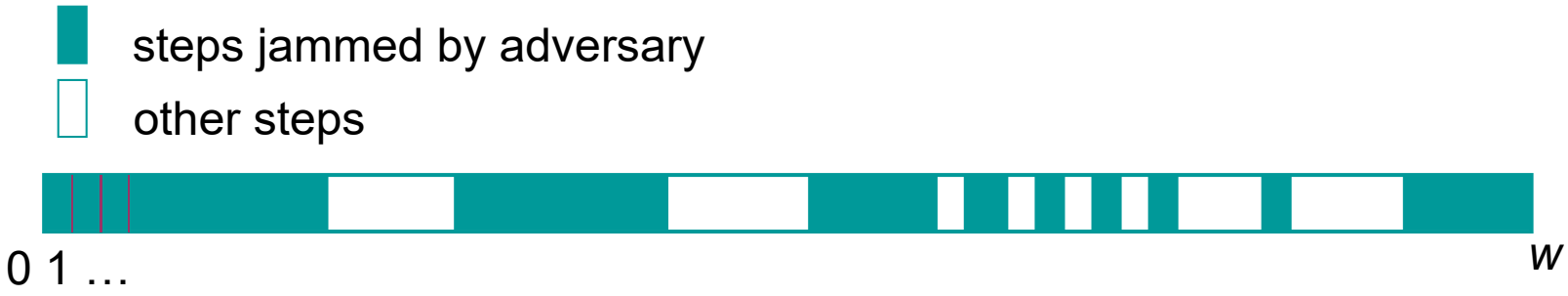
- (T,λ)-bounded adversary, $0 < \lambda < 1$: in any time window of size $w \geq T$, the adversary can jam $\leq \lambda w$ time steps



Adaptive Adversary

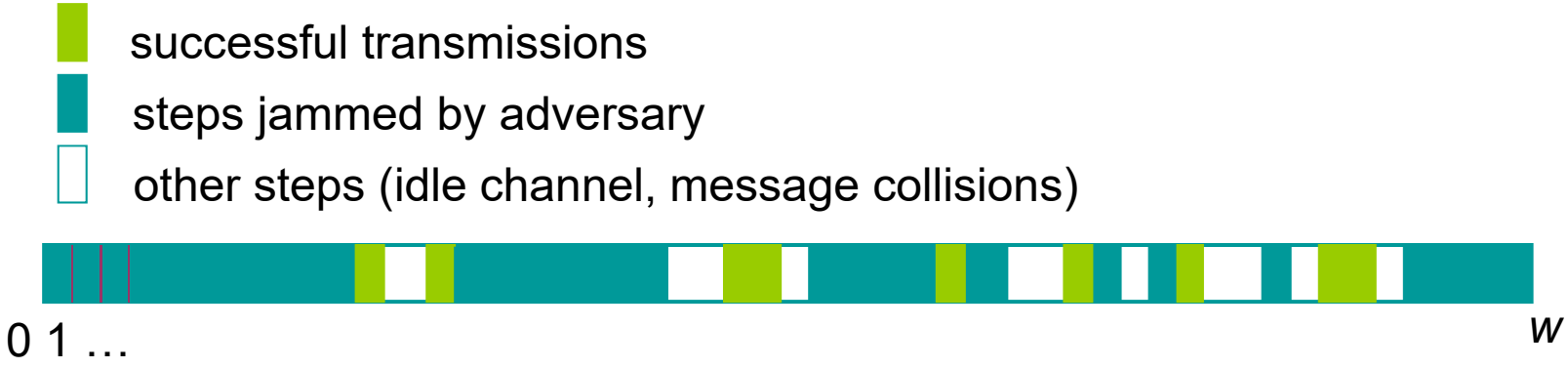
Types of adversarial behavior:

- non-reactive adversary: jamming decision independent of current time step
- reactive adversary: adversary can distinguish between an idle and a busy channel (successful transmission or message collision) at the current time step and base its jamming decision on that



Competitive Analysis

- a protocol is called c-competitive against a (non-) reactive (T,λ)-bounded adversary if the nodes manage to perform successful transmissions in at least a c-fraction of the non-jammed steps (w.h.p. or on expectation), for any sufficiently large number of steps



Main Result

Theorem 6.1: There is a symmetric local-control MAC protocol that is

- constant-competitive against any $(T,1-\varepsilon)$-bounded adversary after $\tilde{\Omega}(T/\varepsilon)$ steps w.h.p., for any constant $0<\varepsilon<1$ and any T,
- energy efficient in a sense that it converges to bounded energy consumption due to message transmissions by nodes under continuous adversarial jamming ($\varepsilon=0$), and
- recovers quickly from any state.

Traditional Defenses

- spread spectrum: frequency hopping over a wide frequency band
  - hard for a jammer to detect the used frequency fast enough in order to jam it
  - Problem: commonly used wireless devices (e.g., 802.11) have relatively narrow frequency bands
- random backoff:
  - adaptive adversary too powerful for MAC protocols based on random backoff or tournaments (including the standard MAC protocol of 802.11)

Simple idea

- each node v sends a message at current time step with probability $p_v \leq p_{max}$, for constant $0 < p_{max} << 1$.
  p $= \sum p_v$ (cumulative probability)
  q_{idle} = probability the channel is idle
  $q_{success}$ = probability that only one node is transmitting (successful transmission)

Claim 6.2: $q_{idle} \cdot p \leq q_{success} \leq (q_{idle} \cdot p)/(1-p_{max})$

Proof: Exercise.

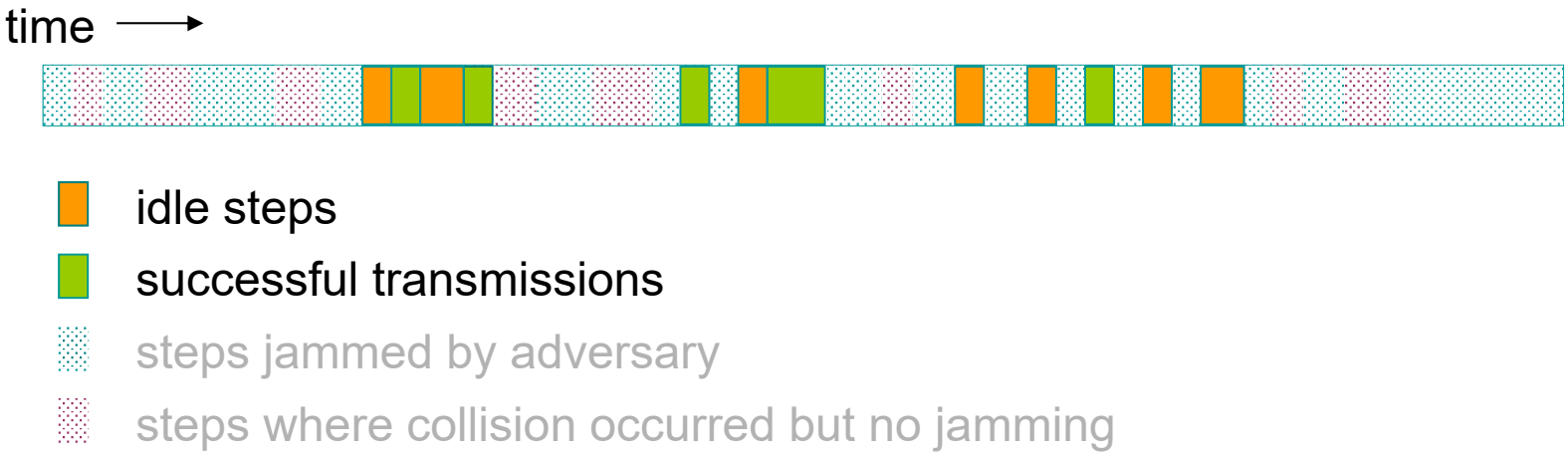
Thus, if the number of times the channel is idle is equal to the number of successful transmissions, then $p = \theta(1)$!

Basic Approach

- a node v adapts p_v based only on steps when an idle channel or a successful message transmission are observed, ignoring all other steps (including all the blocked steps when the adversary jams)

Basic Approach

- a node v adapts p_v based only on steps when an idle channel or a successful message transmission are observed, ignoring all other steps (including all the blocked steps when the adversary jams)



Naïve Protocol

Each time step:

- Node v sends a message with probability p_v . If v does not send a message then
  - if wireless channel is idle then $p_v = (1+\gamma)\, p_v$
  - if v received a message then $p_v = p_v/(1+\gamma)$

Problems

Basic problem: Cumulative probability *p* could be too large.

- almost all time steps blocked due to message collisions

Problems

Basic problem: Cumulative probability *p* could be too large.

- almost all time steps blocked due to message collisions

Problems

Basic problem: Cumulative probability p too large.
$\rightarrow$ almost all time steps blocked due to message collisions

Idea: If more than T consecutive time steps without successful transmissions, then reduce probabilities, which results in fast recovery of p.

Problem: Nodes do not know T. How to learn a good time window threshold?

It turns out that additive-increase additive-decrease is the right strategy!

MAC Protocol

- each node v maintains
  - probability value p_v ,
  - time window threshold T_v , and
  - counter c_v
- Initially, $T_v = c_v = 1$ and $p_v = p_{max}$ (< 1/24).
- synchronized time steps (for ease of explanation)

- Idea: decrease T_v whenever there is a successful transmission but wait for an entire time window (according to current estimate T_v) until you can increase T_v

MAC Protocol

In each step:

- node v sends a message with probability p_v . If v decides not to send a message then
  - if v senses an idle channel, then $p_v = \min\{(1+\gamma)p_v, p_{max}\}$
  - if v successfully receives a message, then $p_v = p_v/(1+\gamma)$ and $T_v = \max\{T_v - 1, 1\}$
- $c_v = c_v + 1$. If $c_v > T_v$ then
  - $c_v = 1$
  - if v did not receive a message successfully in the last T_v steps then $p_v = p_v/(1+\gamma)$ and $T_v = T_v + 1$

We need $\gamma = O(1/(\log T + \text{loglog } n))$ for the protocol to satisfy Theorem 6.1.

Pros and Cons

Pros:

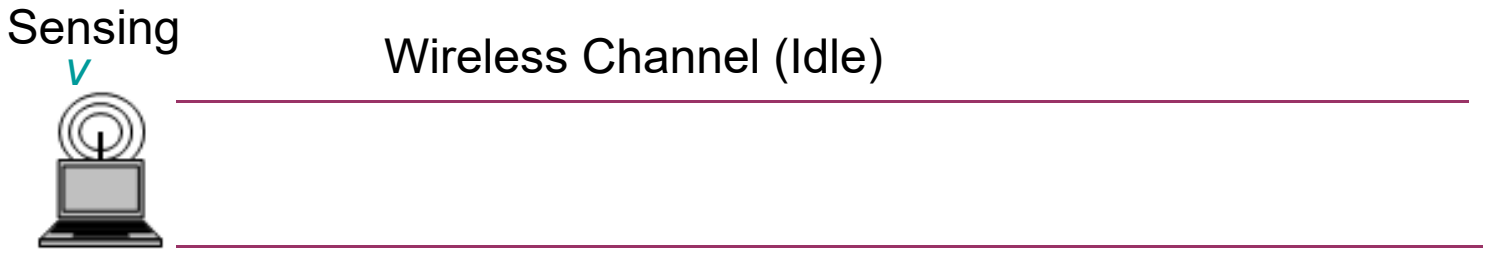
- no prior knowledge of global parameters
  - nodes do not know ε
- no IDs needed

Cons:

- nodes know common rough estimate
  $$\gamma = O(1/(\log T + \log\log n))$$
  - allows for superpolynomial change in n and polynomial change in T over time
- fair channel use is not guaranteed

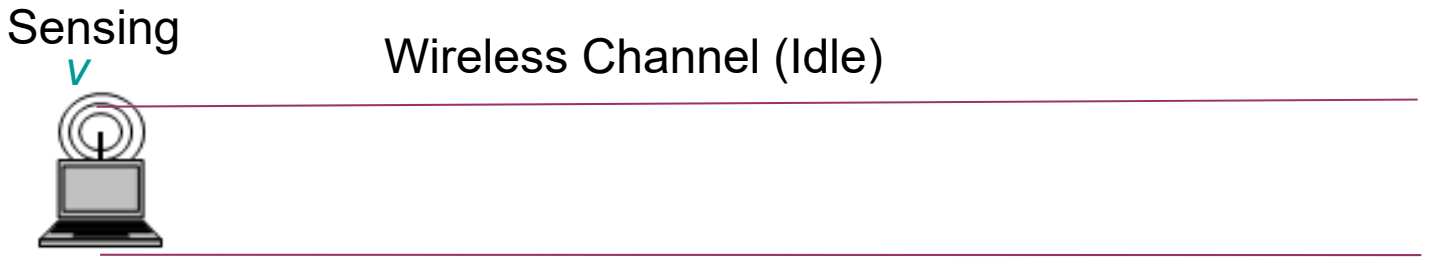
Example: Low value of *p*

- $p_v = 1/n^2$, $T_v = 3$, $c_v = 1$



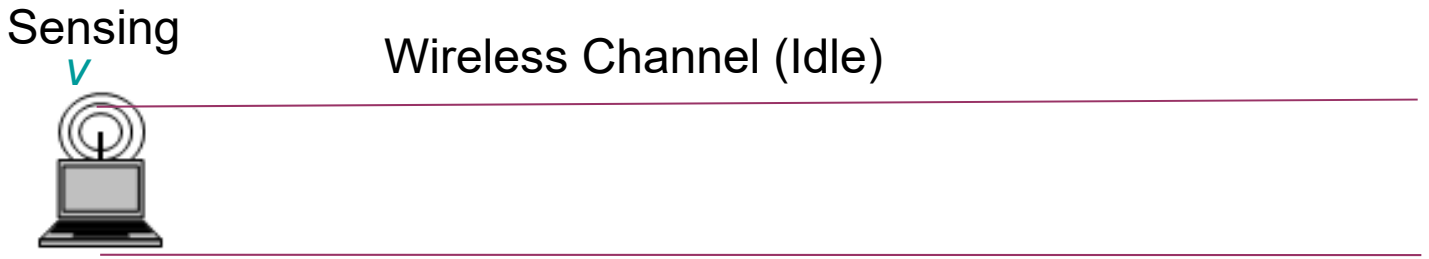
Example: Low value of *p*

- $p_v = (1+\gamma)/n^2$, $T_v = 3$, $c_v = 2$



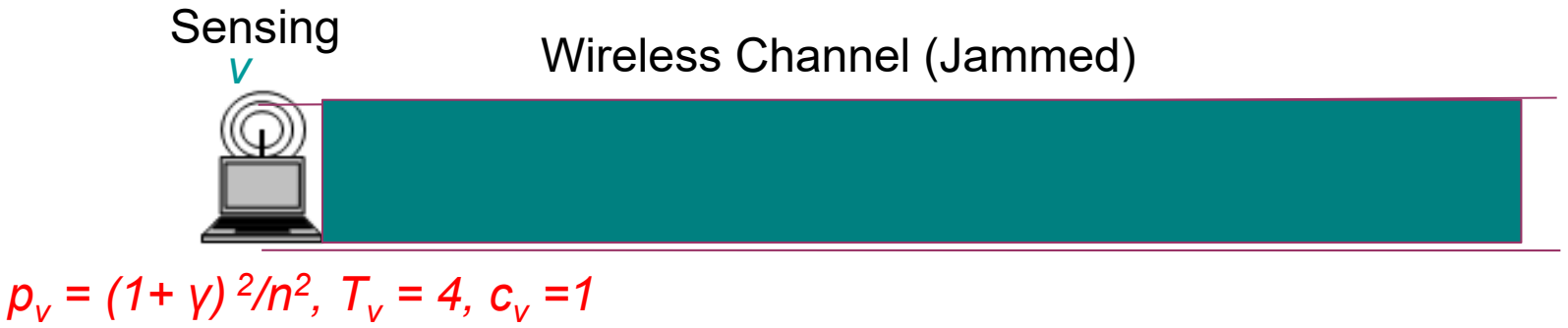
Example: Low value of *p*

- $p_v = (1+\gamma)^2 / n^2$, $T_v = 3$, $c_v = 3$



Example: Low value of *p*

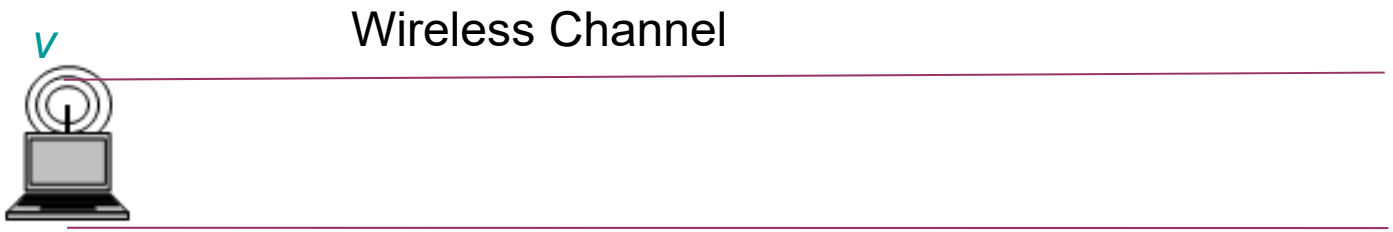
- $p_v = (1+\gamma)^3/n^2$, $T_v = 3$, $c_v = 4$



$p_v = (1+\gamma)^2/n^2$, $T_v = 4$, $c_v = 1$

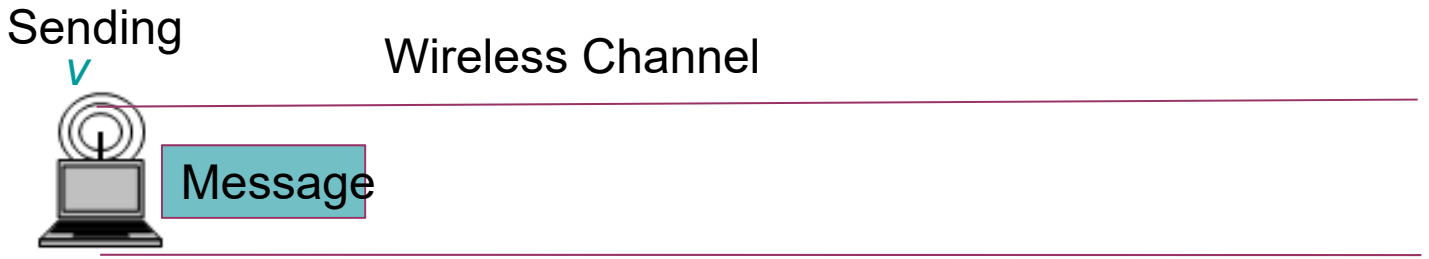
Example: Low value of p

- ~ polylog (n) idle steps later:
  - $p_v \cong c/n$, $T_v \leq \sqrt{T}\ polylog\ (n)$



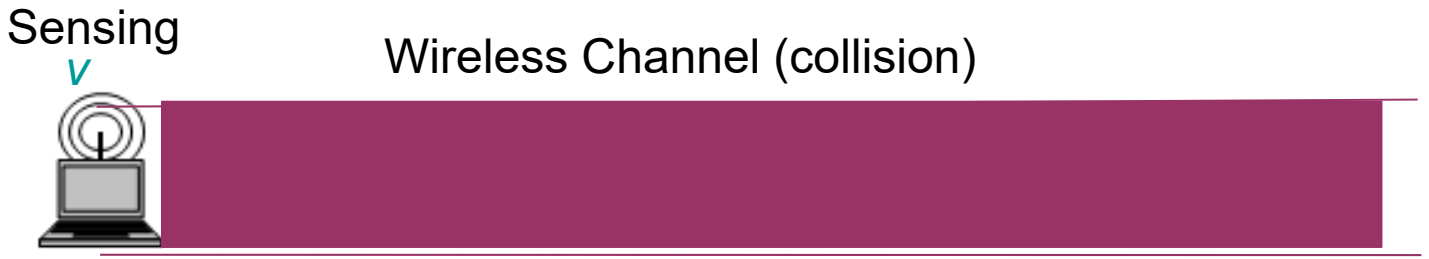
Example: Large p

- $p_v = 1/c$, $T_v = 2$, $c_v = 1$



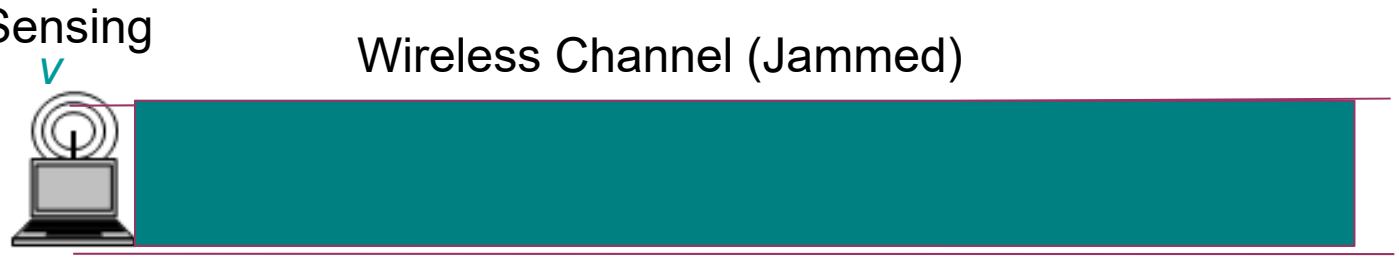
Example: Large *p*

- $p_v = 1/c$, $T_v = 2$, $c_v = 2$



Example: Large *p*

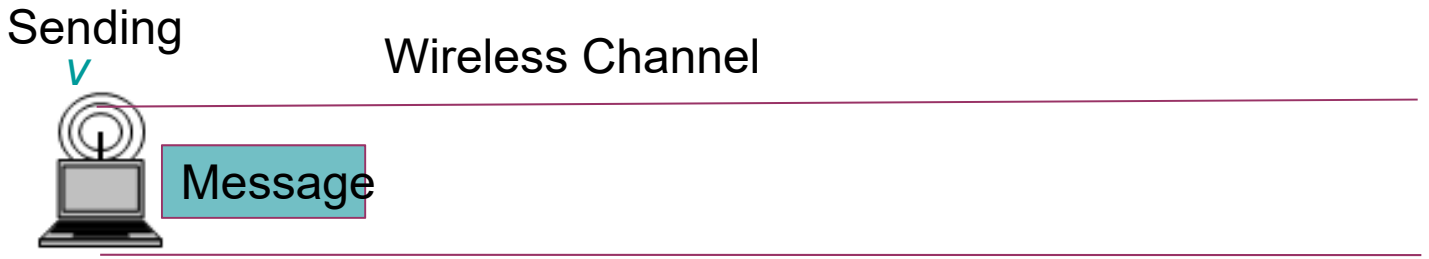
- $p_v = 1/c$, $T_v = 2$, $c_v = 3$



$p_v = 1/[c(1+\gamma)]$, $T_v = 3$, $c_v = 1$

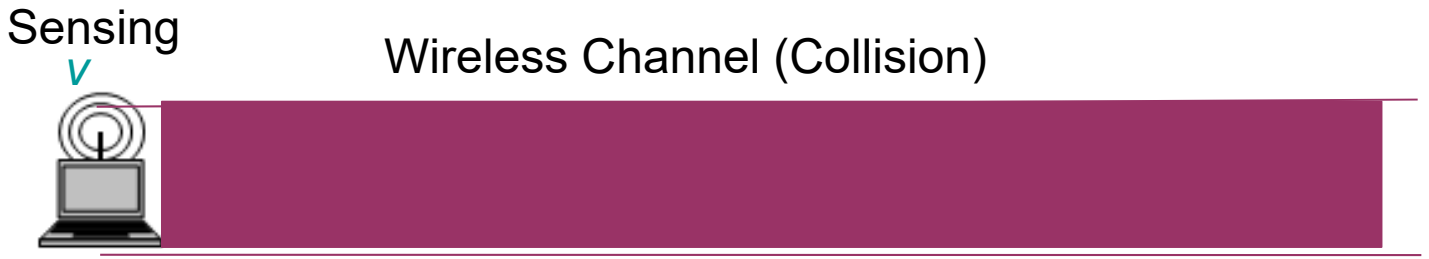
Example: Large *p*

- $p_v = 1/[c(1+\gamma)]$, $T_v = 3$, $c_v = 1$



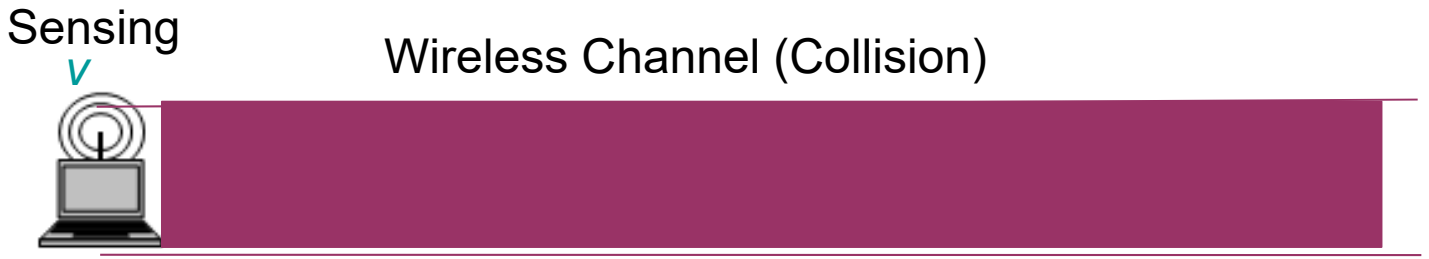
Example: Large *p*

- $p_v = 1/[c(1+\gamma)]$, $T_v = 3$, $c_v = 2$



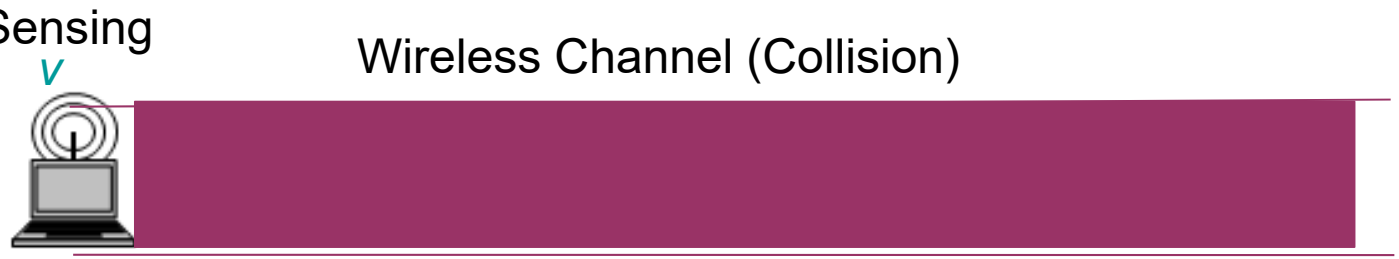
Example: Large *p*

- $p_v = 1/[c(1+\gamma)]$, $T_v = 3$, $c_v = 3$



Example: Large *p*

- $p_v = 1/[c(1+\gamma)]$, $T_v = 3$, $c_v = 4$



$p_v = 1/[c(1+\gamma)^2]$, $T_v = 4$, $c_v = 1$

MAC Protocol

In each step:

- node v sends a message with probability p_v . If v decides not to send a message then
  - if v senses an idle channel, then $p_v = \min\{(1+\gamma)p_v, p_{max}\}$
  - if v successfully receives a message, then $p_v = p_v/(1+\gamma)$ and $T_v = \max\{T_v - 1, 1\}$
- $c_v = c_v + 1$. If $c_v > T_v$ then
  - $c_v = 1$
  - if v did not receive a message successfully in the last T_v steps then $p_v = p_v/(1+\gamma)$ and $T_v = T_v + 1$

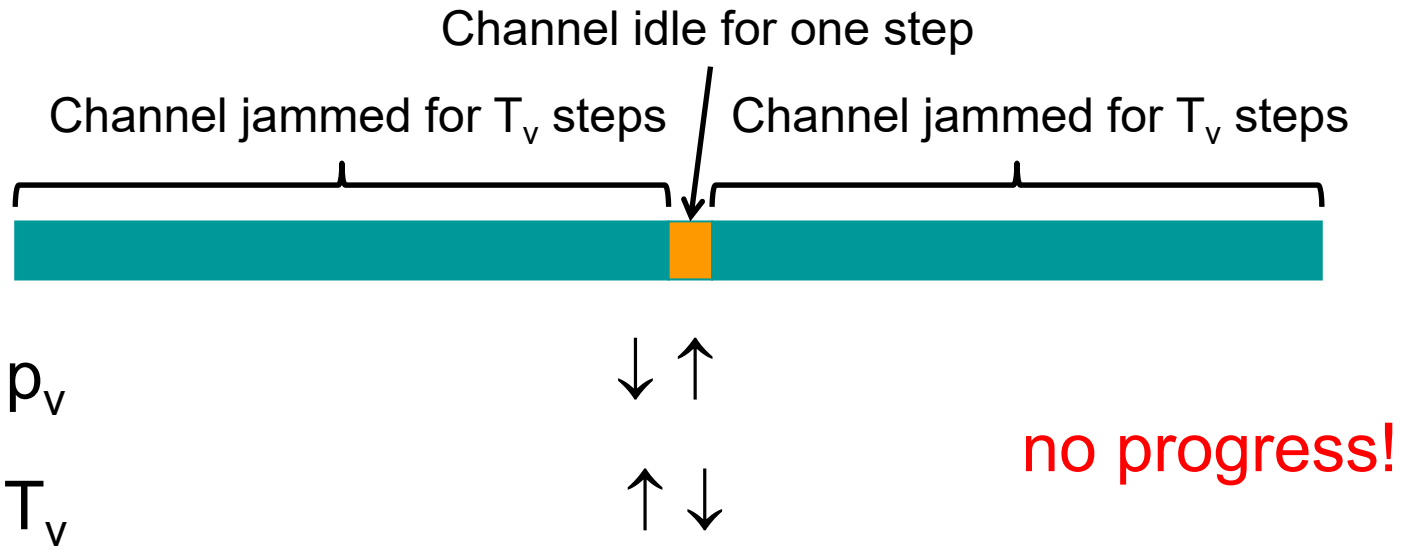
Why not at idle steps?

We need $\gamma = O(1/(\log T + \text{loglog } n))$ for the protocol to satisfy Theorem 6.1.

Counterexample

Suppose that $\Sigma_v\ p_v$ is very low.

Repeat indefinitely:



Proof of Theorem 6.1

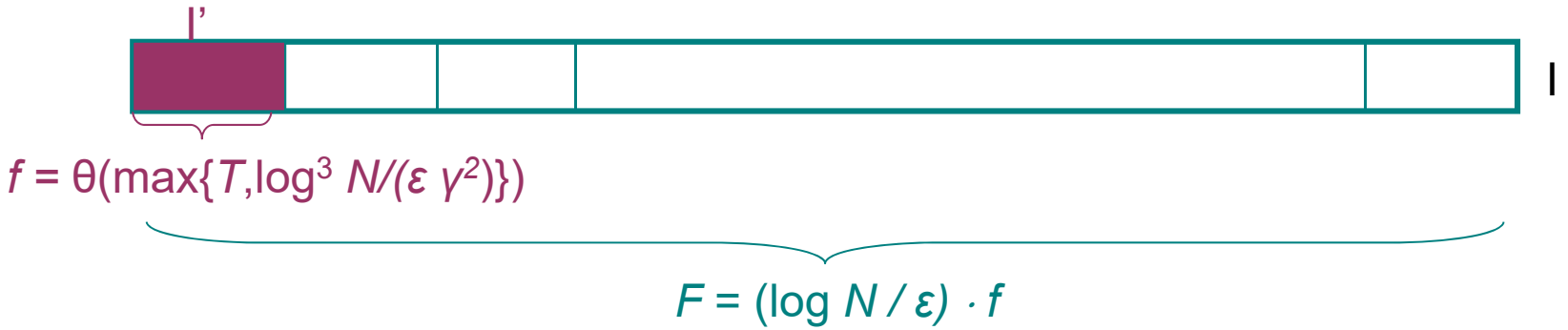
- Let $N = \max\{T,n\}$

Lemma 6.3: The MAC protocol is constant-competitive under any $(T,1-\varepsilon)$-bounded adversary if the protocol is executed for $\Omega(\log N \cdot \max\{T,\log^3 N/(\varepsilon\,\gamma^2)\} / \varepsilon)$ steps w.h.p., for any constant $0<\varepsilon<1$ and any T.

Proof sketch of Lemma 6.3

- Show competitiveness for time frames of $F = \theta((\log N \cdot \max\{T, \log^3 N/(\varepsilon \gamma^2)\} / \varepsilon)$ many steps

  If we can show constant competitiveness for any such time frame of size F, the theorem follows
- We subdivide each frame:



Proof sketch of Lemma 6.3

- $p > 1/(f^2(1+\gamma)^{2\sqrt{f}})$ and $T_v < \sqrt{F}$, in each subframe I' w.h.p.
- $p<12$ and $p>1/12$ within subframe I' with moderate probability (so that adaptive adversarial jamming not successful)
- Constant throughput in I' with moderate probability
- Over a logarithmic number of subframes, constant throughput in frame I of size F w.h.p.

Jade Protocol

Alternative protocol, called Jade, that achieves fairness:

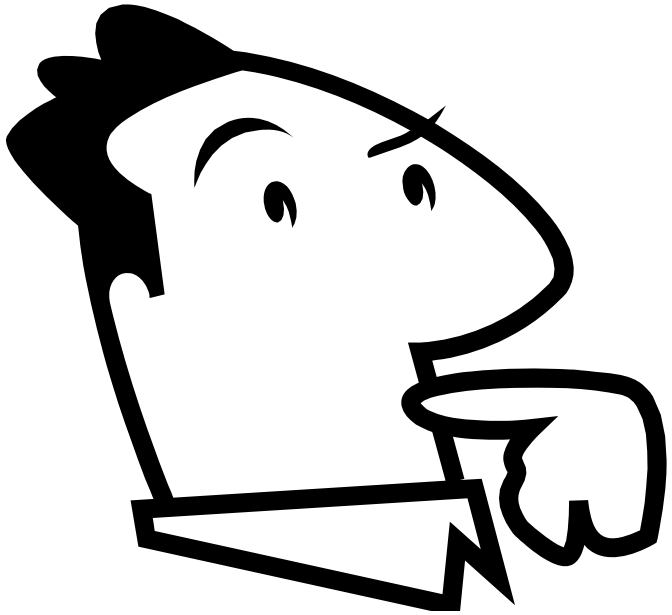
In each step:

- node v sends a message with probability p_v . If v decides not to send a message then
  - if v senses an idle channel, then $p_v = \min\{(1+\gamma)p_v, p_{max}\}$
  - if v successfully receives a message, then $p_v = p_v/(1+\gamma)$ and $T_v = \max\{T_v - 1, 1\}$
- $c_v = c_v + 1$. If $c_v > T_v$ then
  - $c_v = 1$
  - if v only saw busy steps in the last T_v steps then $p_v = p_v/(1+\gamma)$ and $T_v = T_v + 1$

We need $\gamma = O(1/(\log T + \text{loglog } n))$ for the protocol to satisfy Theorem 6.1.

References

- Baruch Awerbuch, Andréa W. Richa, Christian Scheideler: A jamming-resistant MAC protocol for single-hop wireless networks. PODC 2008: 45-54.
- Andréa W. Richa, Christian Scheideler, Stefan Schmid, Jin Zhang: A Jamming-Resistant MAC Protocol for Multi-Hop Wireless Networks. DISC 2010: 179-193.
- Andréa W. Richa, Christian Scheideler, Stefan Schmid, Jin Zhang: Competitive and Fair Medium Access Despite Reactive Jamming. ICDCS 2011: 507-516.
- Andréa W. Richa, Christian Scheideler, Stefan Schmid, Jin Zhang: Competitive and fair throughput for co-existing networks under adversarial interference. PODC 2012: 291-300.
- Adrian Ogierman, Andréa W. Richa, Christian Scheideler, Stefan Schmid, Jin Zhang: Competitive MAC under adversarial SINR. INFOCOM 2014: 2751-2759.



Questions?