Overview

• Best-Effort Broadcast
• (Regular) Reliable Broadcast (2)
• Uniform Reliable Broadcast (2)

• Stubborn Broadcast
• Logged Best-Effort Broadcast
• Logged Uniform Reliable Broadcast

• Probabilistic Broadcast (2)
Abstracting Processes

• Crash-stop
• Omissions
• Crash-recovery
• Eavesdropping faults
• Arbitrary-fault (Byzantine)

• Faulty or correct
Timing Assumptions

- Asynchronous System
- Synchronous System
- Partial Synchrony
Distributed-System Models

- Fail-stop
  - crash-stop, perfekt links, perfect failure detector ($P$)
- Fail-noisy
  - crash-stop, perfekt links, eventually $P$ ($\diamondsuit P$)
- Fail-silent
  - crash-stop, perfekt links, no failure detector
- Fail-recovery
  - crash-recovery, stubborn links, eventual leader detector ($\Omega$)
- Fail-arbitrary
  - fail-arbitrary, authenticated perfekt links
Motivation

- **Client-server scheme** – *point-to-point* communication
  - Useful when *reliable*, e.g. TCP

- Bigger systems usually more than 2 processes
  - *Broadcast* abstractions convenient
    - Send to all processes, in a single one-shot op.

- Reliability req. of p2p not directly transposable
  - “No message lost or duplicated”
  - Complex for broadcast...
Best-Effort Broadcast

• Burden of ensuring reliability on sender:
  – Receivers unconcerned with enforcing reliability
  – No delivery guarantees if sender fails
Best-Effort Broadcast

Module 3.1: Interface and properties of best-effort broadcast

Module:

| Name: BestEffortBroadcast, instance \texttt{beb}. |

Events:

| Request: \texttt{\langle beb, Broadcast \mid m \rangle}: Broadcasts a message \textit{m} to all processes. |
| Indication: \texttt{\langle beb, Deliver \mid p, m \rangle}: Delivers a message \textit{m} broadcast by process \textit{p}. |

Properties:

| BEB1: \textit{Validity}: If a correct process broadcasts a message \textit{m}, then every correct process eventually delivers \textit{m}. |
| BEB2: \textit{No duplication}: No message is delivered more than once. |
| BEB3: \textit{No creation}: If a process delivers a message \textit{m} with sender \textit{s}, then \textit{m} was previously broadcast by process \textit{s}. |
Best-Effort Broadcast

Algorithm 3.1: Basic Broadcast

**Implements:**
BestEffortBroadcast, **instance** \( \text{beb} \).

**Uses:**
PerfectPointToPointLinks, **instance** \( \text{pl} \).

**upon event** \( \langle \text{beb}, \text{Broadcast} \mid m \rangle \) **do**

**forall** \( q \in \Pi \) **do**
**trigger** \( \langle \text{pl}, \text{Send} \mid q, m \rangle \);

**upon event** \( \langle \text{pl}, \text{Deliver} \mid p, m \rangle \) **do**
**trigger** \( \langle \text{beb}, \text{Deliver} \mid p, m \rangle \);
Best-Effort Broadcast

**Figure 3.1:** Sample execution of basic broadcast
(Regular) Reliable Broadcast

• Best-effort only ensures delivery if sender doesn’t crash
  – Processes might not agree on message delivery
  – Even if all messages sent before sender crashes…

• (Regular) Reliable broadcast provides stronger notion of reliability:
  – Ensures agreement even if sender fails.
  – Sender failure – no process delivers message
(Regular) Reliable Broadcast

Module 3.2: Interface and properties of (regular) reliable broadcast

Module:

Name: ReliableBroadcast, instance rb.

Events:

Request: \( \langle \text{rb}, \text{Broadcast} \mid m \rangle \): Broadcasts a message \( m \) to all processes.

Indication: \( \langle \text{rb}, \text{Deliver} \mid p, m \rangle \): Delivers a message \( m \) broadcast by process \( p \).

Properties:

RB1: Validity: If a correct process \( p \) broadcasts a message \( m \), then \( p \) eventually delivers \( m \).

RB2: No duplication: No message is delivered more than once.

RB3: No creation: If a process delivers a message \( m \) with sender \( s \), then \( m \) was previously broadcast by process \( s \).

RB4: Agreement: If a message \( m \) is delivered by some correct process, then \( m \) is eventually delivered by every correct process.
(Regular) Reliable Broadcast

Algorithm 3.2: Lazy Reliable Broadcast

Implements:
ReliableBroadcast, instance rb.

Uses:
BestEffortBroadcast, instance beb;
PerfectFailureDetector, instance \( \mathcal{P} \).

upon \( rb, Init \) do
\[
\text{correct} := \Pi;
\text{from}[p] := [\emptyset]^N;
\]

upon \( rb, Broadcast | m \) do
\[
\text{trigger} \langle \text{beb}, Broadcast | [\text{DATA}, \text{self}, m] \rangle;
\]
(Regular) Reliable Broadcast

\begin{align*}
\textbf{upon event} \langle \text{beb, Deliver} \mid p, [\text{DATA}, s, m] \rangle \textbf{ do} \\
\quad \text{if } m \not\in \text{from}[s] \textbf{ then} \\
\quad \quad \textbf{trigger} \langle \text{rb, Deliver} \mid s, m \rangle; \\
\quad \quad \text{from}[s] := \text{from}[s] \cup \{m\}; \\
\quad \quad \textbf{if } s \not\in \text{correct} \textbf{ then} \\
\quad \quad \quad \textbf{trigger} \langle \text{beb, Broadcast} \mid [\text{DATA}, s, m] \rangle; \\
\end{align*}

\begin{align*}
\textbf{upon event} \langle \mathcal{P}, \text{Crash} \mid p \rangle \textbf{ do} \\
\quad \text{correct} := \text{correct} \setminus \{p\}; \\
\quad \forall m \in \text{from}[p] \textbf{ do} \\
\quad \quad \textbf{trigger} \langle \text{beb, Broadcast} \mid [\text{DATA}, p, m] \rangle;
\end{align*}
(Regular) Reliable Broadcast

Algorithm 3.3: Eager Reliable Broadcast

Implements:

- ReliableBroadcast, instance rb.

Uses:

- BestEffortBroadcast, instance beb.

upon event (rb, Init) do
  delivered := ∅;

upon event (rb, Broadcast | m) do
  trigger (beb, Broadcast | [DATA, self, m]);

upon event (beb, Deliver | p, [DATA, s, m]) do
  if m ∉ delivered then
    delivered := delivered ∪ {m};
    trigger (rb, Deliver | s, m);
    trigger (beb, Broadcast | [DATA, s, m]);
(Regular) Reliable Broadcast

Figure 3.2: Sample execution of reliable broadcast with faulty sender
(Regular) Reliable Broadcast

- Problem: only requires the *correct* processes deliver the same set of messages

Figure 3.3: Nonuniform reliable broadcast
Uniform Reliable Broadcast

Module 3.3: Interface and properties of uniform reliable broadcast

Module:

**Name:** UniformReliableBroadcast, instance urb.

**Events:**

**Request:** \( \langle \text{urb}, \text{Broadcast} \mid m \rangle \): Broadcasts a message \( m \) to all processes.

**Indication:** \( \langle \text{urb}, \text{Deliver} \mid p, m \rangle \): Delivers a message \( m \) broadcast by process \( p \).

**Properties:**

**URB1–URB3:** Same as properties RB1–RB3 in (regular) reliable broadcast (Module 3.2).

**URB4: Uniform agreement:** If a message \( m \) is delivered by some process (whether correct or faulty), then \( m \) is eventually delivered by every correct process.

**Different!**
Uniform Reliable Broadcast

Algorithm 3.4: All-Ack Uniform Reliable Broadcast

Implements:
- UniformReliableBroadcast, instance urb.

Uses:
- BestEffortBroadcast, instance beb.
- PerfectFailureDetector, instance \( \mathcal{P} \).

upon event \( \langle \text{urb}, \text{Init} \rangle \) do
- delivered := \( \emptyset \);
- pending := \( \emptyset \);
- correct := \( \Pi \);
- forall \( m \) do \( \text{ack}[m] := \emptyset \);

upon event \( \langle \text{urb}, \text{Broadcast} \mid m \rangle \) do
- pending := pending \( \cup \) \{ (self, m) \};
- trigger \( \langle \text{beb}, \text{Broadcast} \mid \text{DATA}, \text{self}, m \rangle \);
Uniform Reliable Broadcast

upon event \( \langle \text{beb}, \text{Deliver} \mid p, [\text{DATA}, s, m] \rangle \) do
\[
\begin{align*}
& \text{ack}[m] := \text{ack}[m] \cup \{p\}; \\
& \text{if} \ (s, m) \notin \text{pending} \ \text{then} \\
& \qquad \text{pending} := \text{pending} \cup \{(s, m)\}; \\
& \qquad \text{trigger} \ \langle \text{beb}, \text{Broadcast} \mid [\text{DATA}, s, m] \rangle;
\end{align*}
\]

upon event \( \langle \mathcal{P}, \text{Crash} \mid p \rangle \) do
\[
\text{correct} := \text{correct} \setminus \{p\};
\]

function \( \text{candeler}(m) \) returns Boolean is
\[
\begin{align*}
& \text{return} \ (\text{correct} \subseteq \text{ack}[m]);
\end{align*}
\]

upon exists \( (s, m) \in \text{pending} \) such that \( \text{candeler}(m) \land m \notin \text{delivered} \) do
\[
\begin{align*}
& \text{delivered} := \text{delivered} \cup \{m\}; \\
& \text{trigger} \ \langle \text{urb}, \text{Deliver} \mid s, m \rangle;
\end{align*}
\]
Uniform Reliable Broadcast

Figure 3.4: Sample execution of all-ack uniform reliable broadcast
Uniform Reliable Broadcast

• Requires $N > 2f$

**Algorithm 3.5: Majority-Ack Uniform Reliable Broadcast**

**Implements:**
- UniformReliableBroadcast, *instance urb*.

**Uses:**
- BestEffortBroadcast, *instance beb*.

// Except for the function `canddeliver(·)` below and for the absence of `Crash` events
// triggered by the perfect failure detector, it is the same as Algorithm 3.4.

```plaintext
function canddeliver(m) returns Boolean is
  return #(ack[m]) > N/2;
```

Size of ack[m]
Stubborn Broadcast

Module 3.4: Interface and properties of stubborn best-effort broadcast

Module:

**Name:** StubbornBestEffortBroadcast, **instance** sbeb.

**Events:**

**Request:** \( \langle \text{sbeb}, \text{Broadcast} \mid m \rangle \): Broadcasts a message \(m\) to all processes.

**Indication:** \( \langle \text{sbeb}, \text{Deliver} \mid p, m \rangle \): Delivers a message \(m\) broadcast by process \(p\).

**Properties:**

**SBEB1:** Best-effort validity: If a process that never crashes broadcasts a message \(m\), then every correct process delivers \(m\) an infinite number of times.

**SBEB2:** No creation: If a process delivers a message \(m\) with sender \(s\), then \(m\) was previously broadcast by process \(s\).

No duplication gone!
Algorithm 3.6: Basic Stubborn Broadcast

Implements:
StubbornBestEffortBroadcast, instance \textit{sbeb}.

Uses:
StubbornPointToPointLinks, instance \textit{sl}.

\begin{verbatim}
upon event \langle sbeb, Recovery \rangle do
  // do nothing

upon event \langle sbeb, Broadcast | m \rangle do
  forall q \in \Pi do
    trigger \langle sl, Send | q, m \rangle;

upon event \langle sl, Deliver | p, m \rangle do
  trigger \langle sbeb, Deliver | p, m \rangle;
\end{verbatim}
Logged Best-Effort Broadcast

- First for Fail-recovery model
- Strongest model, uniform reliable, not enough
- Difficulty: crashing, recovery and never crashing again is *correct*
- Solution: stable storage, as seen in “logged perfect links"
Logged Best-Effort Broadcast

Module 3.5: Interface and properties of logged best-effort broadcast

Module:

Name: LoggedBestEffortBroadcast, instance lbeb.

Events:

Request: \( lbeb, \text{Broadcast} \mid m \): Broadcasts a message \( m \) to all processes.

Indication: \( lbeb, \text{Deliver} \mid delivered \): Notifies the upper layer of potential updates to variable \( delivered \) in stable storage (which log-delivers messages according to the text).

Properties:

LBEB1: Validity: If a process that never crashes broadcasts a message \( m \), then every correct process eventually log-delivers \( m \).

LBEB2: No duplication: No message is log-delivered more than once.

LBEB3: No creation: If a process log-delivers a message \( m \) with sender \( s \), then \( m \) was previously broadcast by process \( s \).
Logged Best-Effort Broadcast

Algorithm 3.7: Logged Basic Broadcast

Implements:
\[ \text{LoggedBestEffortBroadcast, instance } lbeb. \]

Uses:
\[ \text{StubbornPointToPointLinks, instance } sl. \]

\begin{align*}
\text{upon event } \{ lbeb, \text{Init} \} \text{ do } \\
& \quad \text{delivered := } \emptyset; \\
& \quad \text{store}(\text{delivered}); \\
\end{align*}

\begin{align*}
\text{upon event } \{ lbeb, \text{Recovery} \} \text{ do } \\
& \quad \text{retrieve}(\text{delivered}); \\
& \quad \text{trigger } \{ lbeb, \text{Deliver } | \text{delivered} \}; \\
\end{align*}

\begin{align*}
\text{upon event } \{ lbeb, \text{Broadcast } | m \} \text{ do } \\
& \quad \text{forall } q \in II \text{ do } \\
& \quad \quad \text{trigger } \{ sl, \text{Send } | q, m \}; \\
\end{align*}

\begin{align*}
\text{upon event } \{ sl, \text{Deliver } | p, m \} \text{ do } \\
& \quad \text{if } (p, m) \notin \text{delivered} \text{ then } \\
& \quad \quad \text{delivered := delivered } \cup \{ (p, m) \}; \\
& \quad \quad \text{store}(\text{delivered}); \\
& \quad \quad \text{trigger } \{ lbeb, \text{Deliver } | \text{delivered} \}; \\
\end{align*}
Logged Uniform Reliable Broadcast

Module 3.6: Interface and properties of logged uniform reliable broadcast

Module:

Name: LoggedUniformReliableBroadcast, instance lurb.

Events:

Request: \( lurb, \text{Broadcast} | m \): Broadcasts a message \( m \) to all processes.

Indication: \( lurb, \text{Deliver} | \text{delivered} \): Notifies the upper layer of potential updates to variable \( \text{delivered} \) in stable storage (which log-delivers messages according to the text).

Properties:

LURB1–LURB3: Same as properties LBEB1–LBEB3 in logged best-effort broadcast (Module 3.5).

LURB4: Uniform agreement: If a message \( m \) is log-delivered by some process (whether correct or faulty), then \( m \) is eventually log-delivered by every correct process.
Logged Uniform Reliable Broadcast

Algorithm 3.8: Logged Majority-Ack Uniform Reliable Broadcast

Implements:
LoggedUniformReliableBroadcast, instance lurb.

Uses:
StubbornBestEffortBroadcast, instance sbeb.

upon event ⟨ lurb, Init ⟩ do
  delivered := ∅;
  pending := ∅;
  forall m do ack[m] := ∅;
  store(pending, delivered);

upon event ⟨ lurb, Recovery ⟩ do
  retrieve(pending, delivered);
  trigger ⟨ lurb, Deliver | delivered ⟩;
  forall (s, m) ∈ pending do
    trigger ⟨ sbeb, Broadcast | [DATA, s, m] ⟩;
Logged Uniform Reliable Broadcast

upon event (lurb, Broadcast | m) do
  pending := pending ∪ {(self, m)};
  store(pending);
  trigger (sbeb, Broadcast | [DATA, self, m]);

upon event (sbeb, Deliver | p, [DATA, s, m]) do
  if (s, m) \notin pending then
    pending := pending ∪ {(s, m)};
    store(pending);
    trigger (sbeb, Broadcast | [DATA, s, m]);
  if p \notin ack[m] then
    ack[m] := ack[m] ∪ {p};
    if #(ack[m]) > N/2 ∧ (s, m) \notin delivered then
      delivered := delivered ∪ {(s, m)};
      store(delivered);
      trigger (lurb, Deliver | delivered);
Probabilistic Broadcast

- No deterministic broadcast guarantees
- Offers “cost” reduction at the price of lower reliability
  a. Reliability not scalable – *ack implosion* problem
  b. Possible solution – requires configuration

*Figure 3.5: Direct vs. hierarchical communication for sending messages and receiving acknowledgments*

- *Epidemic dissemination* – rumor spreading, gossiping
Probabilistic Broadcast

Module 3.7: Interface and properties of probabilistic broadcast

Module:

Name: ProbabilisticBroadcast, instance pb.

Events:

Request: \( \langle pb, \text{Broadcast} \mid m \rangle \): Broadcasts a message \( m \) to all processes.

Indication: \( \langle pb, \text{Deliver} \mid p, m \rangle \): Delivers a message \( m \) broadcast by process \( p \).

Properties:

PB1: Probabilistic validity: There is a positive value \( \varepsilon \) such that when a correct process broadcasts a message \( m \), the probability that every correct process eventually delivers \( m \) is at least \( 1 - \varepsilon \).

PB2: No duplication: No message is delivered more than once.

PB3: No creation: If a process delivers a message \( m \) with sender \( s \), then \( m \) was previously broadcast by process \( s \).
Eager Probabilistic Broadcast

- Sends to $k$ random processes – the \textit{fanout}
- A \textit{round of gossiping} = receiving and resending message
- $R$ rounds of gossiping per message
- $R$ and $k$ determine efficiency of algorithm

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure3_6.pdf}
\caption{Epidemic dissemination or gossip (with fanout 3)}
\end{figure}
Eager Probabilistic Broadcast

Algorithm 3.9: Eager Probabilistic Broadcast

Implements:
ProbabilisticBroadcast, instance pb.

Uses:
FairLossPointToPointLinks, instance fll.

upon event \langle pb, Init \rangle do
  delivered := \emptyset;

procedure gossip(msg) is
  forall t \in picktargets(k) do trigger \langle fll, Send | t, msg \rangle;

upon event \langle pb, Broadcast | m \rangle do
  delivered := delivered \cup \{m\};
  trigger \langle pb, Deliver | self, m \rangle;
  gossip([GOSSIP, self, m, R]);

upon event \langle fll, Deliver | p, [GOSSIP, s, m, r] \rangle do
  if m \notin delivered then
    delivered := delivered \cup \{m\};
    trigger \langle pb, Deliver | s, m \rangle;
  if r > 1 then gossip([GOSSIP, s, m, r - 1]);
Lazy Probabilistic Broadcast

- EPB too eager; consumes resources and causes redundant transmissions

1. Gossip until e.g. N/2 processes infected (push-phase)
2. Missed processes ask for message (pull-phase)
Lazy Probabilistic Broadcast

Algorithm 3.10: Lazy Probabilistic Broadcast (part 1, data dissemination)

*Implements:*  
ProbabilisticBroadcast, instance *pb*.

*Uses:*  
FairLossPointToPointLinks, instance *fll*;  
ProbabilisticBroadcast, instance *upb*. // an unreliable implementation

upon event ⟨ *pb*, *Init* ⟩ do  
next := [1]^N;  
*lsn* := 0;  
*pending* := ∅; *stored* := ∅;

procedure *gossip*(msg) is  
forall *t* ∈ picktargets(*k*) do trigger ⟨ *fll*, Send | *t*, msg ⟩;

upon event ⟨ *pb*, Broadcast | *m* ⟩ do  
*lsn* := *lsn* + 1;  
trigger ⟨ *upb*, Broadcast | [DATA, *self*, *m*, *lsn*] ⟩;
Lazy Probabilistic Broadcast

upon event \( \langle \text{upb}, \text{Deliver} \mid p, [\text{DATA, } s, m, sn] \rangle \) do

if \( \text{random}([0, 1]) > \alpha \) then

\( \text{stored := stored} \cup \{[\text{DATA, } s, m, sn]\} \);

if \( sn = \text{next}[s] \) then

\( \text{next}[s] := \text{next}[s] + 1; \)

\( \text{trigger} \langle \text{pb, Deliver} \mid s, m \rangle; \)

else if \( sn > \text{next}[s] \) then

\( \text{pending := pending} \cup \{[\text{DATA, } s, m, sn]\} \);

\( \text{forall } \text{missing} \in [\text{next}[s], \ldots, sn - 1] \) do

if no \( m' \) exists such that \([\text{DATA, } s, m', \text{missing}] \in \text{pending}\) then

\( \text{gossip}([\text{REQUEST, self, } s, \text{missing}, R - 1]); \)

\( \text{starttimer}(\Delta, s, sn); \)
Lazy Probabilistic Broadcast

\[
\text{upon event } \langle \text{fl}, \text{Deliver} | p, [\text{REQUEST}, q, s, sn, r] \rangle \text{ do}
\]
\[
\text{if exists } m \text{ such that } [\text{DATA}, s, m, sn] \in \text{stored then}
\]
\[
\text{trigger } \langle \text{fl}, \text{Send} | q, [\text{DATA}, s, m, sn] \rangle;
\]
\[
\text{else if } r > 0 \text{ then}
\]
\[
gossip([\text{REQUEST}, q, s, sn, r - 1]);
\]

\[
\text{upon event } \langle \text{fl}, \text{Deliver} | p, [\text{DATA}, s, m, sn] \rangle \text{ do}
\]
\[
pending := pending \cup \{[\text{DATA}, s, m, sn]\};
\]

\[
\text{upon exists } [\text{DATA}, s, x, sn] \in \text{pending such that } sn = \text{next}[s] \text{ do}
\]
\[
\text{next}[s] := \text{next}[s] + 1;
\]
\[
pending := pending \setminus \{[\text{DATA}, s, x, sn]\};
\]
\[
\text{trigger } \langle \text{pb}, \text{Deliver} | s, x \rangle;
\]

\[
\text{upon event } \langle \text{Timeout} | s, sn \rangle \text{ do}
\]
\[
\text{if } sn > \text{next}[s] \text{ then}
\]
\[
\text{next}[s] := sn + 1;
\]